Conceptos básicos: planificando el juego

En esta zona de la obra abordaremos todo lo concerniente al diseño y programación del juego que va a servir de ejemplo a lo largo de la obra, siempre en concordancia, claro está, con las demás áreas.

Aunque es importante hacer un inciso antes de empezar, por lo que en este primer coleccionable comentaremos los procedimientos básicos que se deben seguir a la hora de abordar la producción de un videojuego.

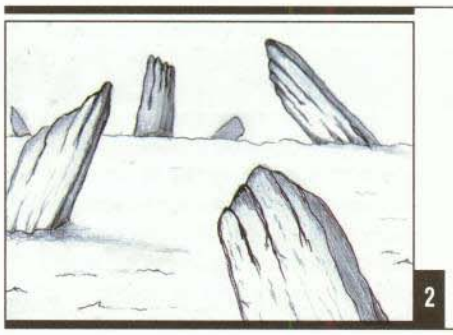
En todo proceso de realización de un proyecto, es fundamental aplicar una cierta disciplina y sobre todo tener organización. Separar las diferentes tareas ayuda, en gran medida, a planificar el trabajo global, permitiendo crear esquemas de resultados en un tiempo prefijado. Y lo que es más importante, facilita enormemente cambios y ampliaciones futuras y corrección de posibles errores.

Hay numerosos sistemas para crear una planificación. Generalmente, se utilizan diagramas o esquemas parecidos a los organigramas de un programa de ordenador o, sin ir más lejos, al esquema de horarios y clases de un estudiante. Nosotros utilizaremos un diagrama de *GANTT* bastante básico, consistente en colocar en un eje vertical los procesos del desarrollo y en uno horizontal el tiempo a emplear en cada uno de ellos.

Los siguientes diagramas de Gantt muestran la evolución del trabajo que realizaremos en esta obra para la terminación de nuestro proyecto. (ver diagramas) En el CD que adjuntamos encontraréis una versión de los diagramas a tamaño folio para colocarlo en la pared, pizarra o tablón de desarrollo.

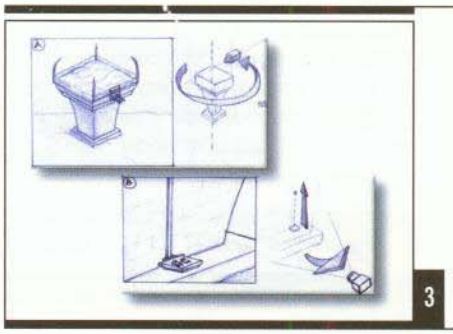
En las primeras seis semanas estaremos prácticamente sólo con el diseño del juego. Haremos un guión de nuestra idea describiendo cuál será el argumento del juego, qué estilo tendrá, cómo serán los protagonistas, las situaciones y la ambientación sonora y gráfica.

Dibujaremos bocetos con el aspecto gráfico de todos los elementos del juego, desde el logotipo, pasando por el menú,



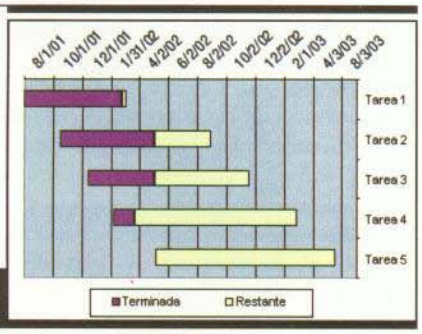
2

Es conveniente dibujar diseños de la forma que tendrá el entorno del juego. Esto ayudará a los grafistas en el modelado y texturizado.



3

El Story-board refleja las fases de la animación en forma de viñetas y permite realizar anotaciones técnicas para los grafistas.



1

El diagrama de GANTT permite planificar las tareas de desarrollo de una manera simple y eficaz.

Diagrama 1

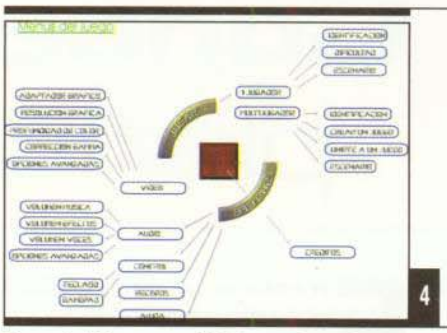
fuentes, personajes, objetos, decorados e incluso los iconos.

Para la presentación y cinemáticas (animaciones) dibujaremos las secuencias en *StoryBoards* y prepararemos dibujos detallados del mapeado por donde transcurrirán las acciones, de esa forma los grafistas podrán ver cómo serán los niveles o cada fase del juego. (Diagrama 2 y 3).

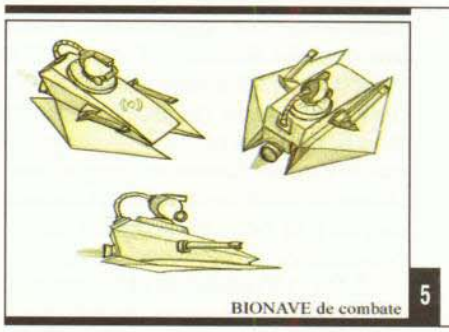
No es totalmente imprescindible cumplir estos diagramas de trabajo al pie de la letra, sobre todo si no tenemos ningún tipo de presión externa, como por ejemplo, la entrega del juego en un tiempo prefijado. Incluso en las grandes desarrolladoras se producen retrasos en sus producciones, muchas veces debido a cambios de personal, problemas de financiación o simplemente a fallos técnicos. Todos recordamos los múltiples retrasos en juegos como Blade (de origen español), Daikatana de John Romero y Collin McRae, etc.

En nuestro caso, es muy normal que cierta rutina se resista a funcionar o que queramos mejorar aún más algunos aspectos gráficos importantes. Sin embargo, no debemos excedernos demasiado; es mejor que nos sobre tiempo a que nos falte. También nos ayudará enormemente a estar preparados para futuros encargos y a consolidar, aún más si cabe, al equipo.

Una cuestión importante es la organización de todo el material resultante en el trabajo diario. Los gráficos, sonidos, diseños y códigos deben de ir almacenados en carpetas perfectamente ordenadas y a los ficheros se les debe asignar nombres coherentes y fáciles de entender. Es conveniente también apuntar los nombres, características y rutas de los archivos de trabajo en papel o en el bloc de notas. A medida que el proyecto avanza, el material va aumentando y sería fácil que se extraviase por los entresijos del disco duro. También es importante separar los ficheros maestros de los resultantes para poder así aplicar modificaciones en cualquier momento del desarrollo o poseer variantes de una misma idea.



El menú del juego permitirá al usuario configurar y elegir el tipo de juego. En su diseño se debe incluir todas las opciones de configuración posibles.



Resulta extremadamente útil para el posterior modelado realizar bocetos del protagonista del juego desde varias vistas.

El modelado del Ford Focus en el juego Colin McRae 3 de la empresa británica Codemaster es asombroso.

Diagrama 2 — Planning del Mes 2

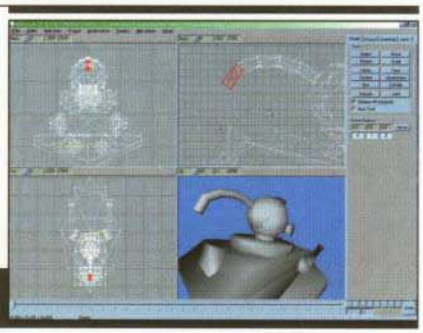
Sección	Semana 5	Semana 6	Semana 7	Semana 8
▶ Grafismo	Título y logotipo	Tipografía	Modelado Bio-naves	Texturizado Bio-naves
▶ Desarrollo Diseño	Diseño del entorno	Presentación, menús y tutorial		
▶ Desarrollo Programación			Editor de zonas de lucha	Esquema del código del juego
▶ Audio		Crear disparos, explosiones y otros SFX	Crear disparos, explosiones y otros SFX	Crear disparos, explosiones y otros SFX

Diagrama 3 — Planning del Mes 3

Sección	Semana 9	Semana 10	Semana 11	Semana 12
▶ Grafismo	Texturizado Bio-naves	Animación Bio-naves	Modelado y animación animales	Modelado y animación plantas
▶ Desarrollo Diseño				
▶ Desarrollo Programación	Visualización del terreno	Indicadores de pantalla	Controlando movimientos del jugador	Controlando acciones del jugador
▶ Audio	Crear entorno, menús y voz en off	Crear entorno, menús y voz en off	Crear entorno, menús y voz en off	Crear la música del menú

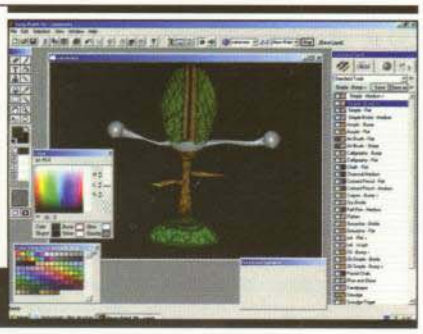
NOTA

La realización de un videojuego implica disciplina y organización en cada parte del desarrollo y sobre todo mucha paciencia.



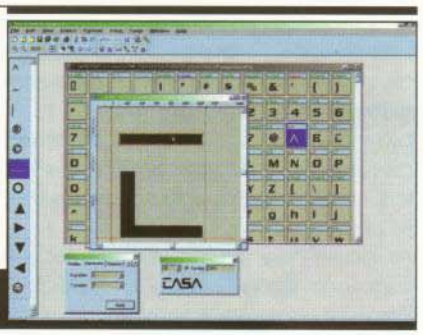
7

El siguiente paso después de dibujar los bocetos es modelar en 3D los objetos. El número de polígonos en cada modelo es crucial para el rendimiento del juego.



8

Acabado el proceso de dar forma a los objetos mediante el modelado se pasa al texturizado, donde el modelo adquiere la apariencia deseada.



9

El diseño de un juego de letras para nuestro videojuego le confiere identidad y personalidad, ayudando a mantener el estilo gráfico general.

Y quizás lo más esencial de todo, y creo que no hace falta mencionarlo, es realizar copias de seguridad (backup) de nuestro trabajo periódicamente. Si disponemos del equipo necesario, se podrían hacer copias históricas de nuestros directorios en CD cada semana o habilitar otras carpetas o discos duros para realizar backups diarios.

Otra cuestión primordial es que cada apartado del desarrollo debe mantener una estrecha relación técnica. El departamento gráfico y sonoro deben saber exactamente qué tipo de ficheros y formatos deben producir, para luego poder ser implementados correctamente y sin problemas en programación. Por eso, en el diseño del juego, es importante discutir qué posibilidades técnicas son viables atendiendo a las herramientas de que se dispone. No sirve, por ejemplo, realizar texturas en formato .JPG cuando el motor del juego sólo puede manipular formatos .BMP; o realizar la música en .MP3 cuando sólo se dispone de reproducción de ficheros .WAV. Es muy fácil cometer errores a la hora de modelar los objetos que irán en un juego 3D. Los grafistas deben de tener en cuenta las posibilidades de rendimiento que posee el motor 3D que utiliza el departamento de programación. Así, una cantidad elevada de polígonos en los objetos e incluso un tamaño elevado en los *tiles* de texturas, hará que el rendimiento del juego baje sustancialmente.

También los guionistas, bocetistas y diseñadores de niveles deben conocer las posibilidades técnicas posibles. Es muy distinta la idea que se tenga de un juego de la forma de llevarlo a la práctica.

Por todo esto, conviene realizar multitud de pruebas técnicas antes de afrontar las tareas, y así sacar el máximo provecho de las herramientas que se tienen y realizar siempre un diseño adecuado a las posibilidades técnicas.

Básicamente, en todo equipo de desarrollo debe existir una coordinación para controlar el flujo de información entre todos los departamentos. Y cada departamento, a su vez, debe también coordinar sus esfuerzos entre sí. Todo esto, unido, contribuirá a que el proyecto adquiera consistencia y llegue a buen fin.

En la época de los primeros juegos para ordenadores domésticos, era común que los proyectos fueran desarrollados por una o dos personas, ya que

Diagrama 4 — Planning del Mes 4

Sección	Semana 13	Semana 14	Semana 15	Semana 16
▶ Grafismo	Texturizado animales y plantas	Modelado Edificaciones	Texturizado Edificaciones	Fabricar texturas entorno
▶ Desarrollo Diseño				
▶ Desarrollo Programación	Controlar colisiones y alteración del entorno	Efectos especiales con partículas	Colocando objetos en el terreno	Comportamiento de seres vivos y entorno (IA)
▶ Audio	Crear música 1 de combate	Crear música 2 de combate	Crear música del final y tabla record	

Diagrama 5 — Planning del Mes 5

Sección	Semana 17	Semana 18	Semana 19	Semana 20
▶ Grafismo	Fabricar texturas FX e items	Dibujar terrenos. Iconos y gráficos para editor	Dibujar iconos y elementos del menú	Gráficos y textos para la ayuda
▶ Desarrollo Diseño				
▶ Desarrollo Programación	Sistema de juego para un jugador	Sistema de juego multijugador	Implementar audio	Implementar menús
▶ Audio				

10

Daikatana de John Romero puede ser un buen o mal juego, pero su mayor fracaso comercial se debió sin duda a los retrasos que sufrió en su desarrollo.

do principalmente a toda la familia. La presencia de esta clase de negocios es muy común en Internet dónde, cada vez, abunda más la venta virtual por catálogo. Los juegos ya no necesitan estar en las tiendas, dentro de cajas; ahora es posible adquirirlos a través de la red inmediatamente, reduciendo costes.

11

Un esquema básico que muestra la división de tareas para el desarrollo de un videojuego: idea, planificación, desarrollo técnico y cierre.

los requisitos para realizar un videojuego no eran tan desbordantes como lo son ahora. Hoy día, cada vez es menor el número de proyectos de mediana calidad que son llevados a cabo por un solo individuo. Generalmente, son equipos formados por varios miembros cada uno dedicado a una disciplina. En los más básicos encontramos a un programador, un grafista y un músico, aunque cada vez aparece más la figura del webmaster, quien será el encargado de llevar el proyecto a Internet.

Este tipo de grupos de desarrollo está incrementándose y determina un potencial en aumento dentro del sector, el cual crece día a día en todo el mundo.

Generalmente, los proyectos de estos grupos van destinados a un segmento lúdico de clase "B", denominándose "value software". Juegos que, sin ser de una gran calidad, encuentran un tipo de mercado de bajo precio, siendo destina-

NOTA

En el desarrollo de un juego, los diseñadores, grafistas y músicos deben conocer perfectamente las posibilidades técnicas del departamento de programación, como rendimiento del motor del juego o formatos de ficheros

Asimismo, muchos de estos grupos nutren la creciente demanda de publicaciones de quiosco. Es muy común que revistas de carácter familiar, en contadas ocasiones, precisen de *remakes* de clásicos -sobre todo aventuras-, puzzles o simples matamarcianos, e incluyan packs de juegos de clase "B".

Sin embargo, incluso, como "value software", estos juegos no encuentran fácil el camino hacia la comercialización. El incesante crecimiento de la competencia, debido fundamentalmente al aumento de producción, ha creado más controles en la calidad de los proyectos.

Se piden muchas más cosas que hace no más de un año, e incluso algunos juegos de esta categoría tienen tanta calidad que llegan a codearse con grandes producciones. Por este motivo, hay que habituarse a los modos de trabajo de las grandes desarrolladoras y cuidar siempre el producto.

Pero no todo son obstáculos. Existen asociaciones, creadas precisamente para buscar mercado a los trabajos de grupos pequeños, que ayudan en todo momento a los desarrolladores noveles proporcionando información, consejos y mucha, mucha esperanza y aliento. En España y Latinoamérica, por ejemplo, tenemos a la Asociación de Desarrolladores Stratos (ww.stratos-ad.com) que sin lugar a dudas es el paso obligado para todo aquel desarrollador que busque una oportunidad.

En esta obra, abordaremos la producción de un videojuego completo de calidad media, perfectamente viable para que lo lleve a cabo una sola persona o varias.

Con paciencia y disciplina, y sobre todo, siguiendo las secciones paso a paso de cada entrega no habrá ningún problema en completar el videojuego; además podrás aprender los pasos y una forma de trabajo para futuros proyectos. Aunque hay muchas formas y técnicas para realizar un juego, hemos procurado utilizar la más rápida y económica que proporcione unos resultados medianamente profesionales.

12

Este esquema muestra un sistema básico de organización para un grupo de desarrollo: departamentos de diseño gráfico, sonido, programación y comercial-marketing.

Conceptos básicos: importancia de los gráficos

Cuando el hombre descubrió la pintura se dio cuenta del poder de comunicación que podían ofrecer las diferentes formas y colores estampadas en la pared de una cueva.

Aquellas expresiones daban a conocer a los individuos que le rodeaban una muestra visual de sus pensamientos. Unas inquietudes que no tenían fin, totalmente dominadas por un rasgo único en el ser humano, la imaginación.

Evidentemente, la calidad de la pintura y la diversidad de colores hacían posible una mayor perfección de aquel lenguaje plástico que el hombre empezaba a utilizar cada vez más para perpetuar su paso por el mundo. Las técnicas mejoraban y los dibujos adquirían una mayor expresión, llegando incluso a simular profundidad y relieve al utilizar tonos claros y oscuros para crear sombras.

Al principio, se pintaba con las manos en paredes y rocas y luego con diferentes utensilios que mojaban en tintes de grasa animal y mezcla de plantas. Se pasó de la piedra a la madera y al barro y se avanzó hasta poder dibujar en papel. Se consiguió plasmar no sólo los pensamientos sino el habla y se dividió esta capacidad en arte y escritura.

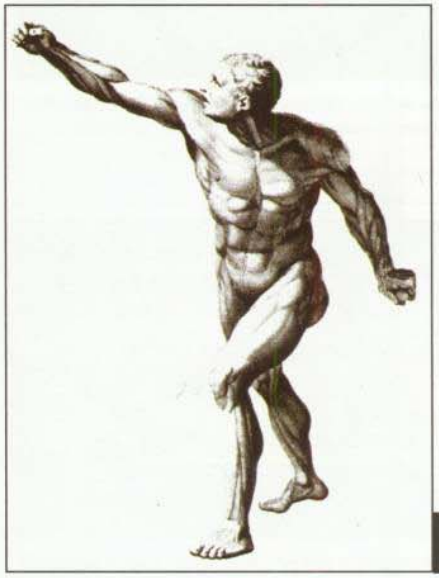
Todo esto nos ayuda a pensar que, en la comunicación, es tan importante la expresión plástica como el habla. El cerebro se nutre constante y principalmente de las sensaciones que llegan a través del ojo. El color, las formas, el movimiento, conforman casi la totalidad de los recuerdos del mundo que nos rodea. Aprovechando todos estos conceptos, el ser humano aprende a comunicarse cada vez más con la Naturaleza y sus semejantes; y lo que es más importante, a valorar y decidir qué es mejor o peor y qué debe prevalecer con más fuerza en el cerebro.

Atendiendo a estas premisas, pensemos por un momento la importancia que tiene, por ejemplo, la elección de colores. Si pintásemos un cuadro, no sería coherente utilizar tonos oscuros y lúgubres en un ambiente primaveral cargado de flores. Es posible hacerlo, pero el resultado final no gustaría a la mayoría de los que observen el cuadro, sólo por el hecho que representa, aunque sí podría llamar la atención. Debemos elegir entonces nosotros mismos qué es lo que de verdad queremos representar y, lo que es más importante, la posible aceptación popular que tendrá.

Pero una elección anormal de colores y luces para un cierto tipo de ambiente en un cuadro puede estar justificada y ser correcta, si ese cuadro está integrado en un conjunto de otras pinturas que conforman una obra común. En definitiva, el conjunto marca la pauta a seguir y sus partes tienen que ir en concordancia

NOTA

El aspecto visual de una idea es la mejor arma para fortalecer los recuerdos y sensaciones. Es muy importante tener muy claro qué se quiere representar exactamente con el dibujo.



1

El efecto de relieve en un dibujo se consigue mediante tonos claros y oscuros.

2

Las texturas, colores e iluminación son fundamentales para conseguir la ambientación deseada.

3

En una aventura gráfica el colorido y el detalle en los gráficos juegan un papel muy importante

4

La n-ésima versión del videojuego Tetris, en esta ocasión para la consola X-Box de Microsoft.

5

Un excesivo colorido en el fondo y en la acción del juego puede llegar a confundir al jugador.

6

La primera impresión de calidad en un videojuego la ofrece los gráficos.

7

Un buen ejemplo de riqueza visual en un juego 3D, el realismo alcanzado es mucho mayor.

con estas pautas para que todo sea coherente y significativo. No tiene sentido, por ejemplo, dibujar en un cómic infantil de humor viñetas con sexo o violencia.

En el mundo de la publicidad, la televisión, el cine, las publicaciones escritas y en nuestro caso, los videojuegos, se tiene siempre en cuenta la capacidad del cerebro de retener imágenes y utilizarlas para decidir, determinar y en definitiva crear un criterio de lo que ve. Se puede decir entonces que, al igual que a nuestros antepasados, la calidad en la elección adecuada de colores, luces y formas harán que nuestras representaciones gráficas tengan o no cabida en los recuerdos de quienes vean nuestra obra.

El diseño gráfico se ha convertido en una disciplina artística plena y, hoy día, los gráficos en un videojuego constituyen unos de los pilares más fuertes en su concepción. Se puede decir que un videojuego con unos gráficos coherentes, cuidados y detallistas tiene asegurado el éxito en un 50 por ciento.

En esta obra vamos a realizar un videojuego en 3D y mostraremos los pasos necesarios para poder dotarlo de grandes dosis de calidad y aproximarnos, lo más posible, a la apariencia profesional que hoy en día se demanda.

IMPORTANCIA DE LOS GRÁFICOS EN UN VIDEOJUEGO

No cabe la menor duda de que el aspecto visual que pueda mostrar un videojuego es lo que primero llama la atención. Es importantísimo pues tener esto en cuenta a la hora de realizar esta parte en el desarrollo de nuestro juego. Los gráficos son la apariencia de nuestra obra y muestran, generalmente, lo que ésta quiere significar, cómo es, de qué trata o a qué estilo pertenece. Una prueba evidente de este hecho, es la elección o no del producto en la tienda sólo con mirar las ilustraciones de las pantallas del juego en la caja; o la obligada inserción de capturas de pantallas o *screenshots* de juegos que se muestran en internet.

Y es un hecho también que, sin conocer nada más, una persona puede juzgar a primera vista un videojuego solamente viendo algunos de sus gráficos o aspectos visuales, aceptándolo o rechazándolo con una simple mirada. Así que unos gráficos llamativos, bien construidos, unos efectos visuales realistas y un movimiento suave y equilibrado pueden hacer que un videojuego llegue a ser aceptado por el usuario sin haber jugado una sola partida en él.

Hacemos una parada para profundizar un poco más en el aspecto de la animación de un gráfico. La finalidad que persigue, por ejemplo, el diseño del protagonista principal de un juego puede truncarse si éste no se mueve con soltura y naturalidad. Si le faltan algunos movimientos necesarios, si se mueve como a trompicones, es decir, si la cinemática de nuestro personaje no es buena, nada podrán hacer unos gráficos impresionantes para salvar nuestro juego del olvido. Hemos de tener en cuenta que todos estos movimientos deben cuidarse al máximo, ya que afectan a la jugabilidad de nuestra creación. En todo caso, el movimiento de nuestros personajes debe casar con la acción de éstos. Si nuestro

NOTA

Por lo general, unos colores vivos y llamativos denotan diversión y un tono infantil.
Por el contrario, unos colores oscuros y con falta de brillantez crean misterio y seriedad.

héroe es un fornido guerrero medieval que porta una espada de grandes dimensiones, podrá caminar erguido, con la cabeza alta, el pecho henchido y paso firme. Pero si la misma espada la carga un lacayo bajito y delgaducho, deberá caminar despacio, encorvado y arrastrando la espada a duras penas.Actualmente, se llegan a utilizar técnicas para el movimiento de personajes muy sofisticadas utilizando aplicaciones especializadas como el Character Studio o capturando movimientos reales con cámaras conectadas al ordenador (sistema motion capture) que luego interpreta y asigna a los modelos virtuales.

Analicemos por un momento aquellos juegos que han marcado historia. La mayoría fueron escritos en épocas en donde la potencia gráfica era pobre y poco potente para representar algo medianamente real, sin embargo, ha sido su diseño, su concepción gráfica, lo que ha ayudado a que permanezca grabado en nuestros pensamientos. Imaginemos, por ejemplo, el legendario juego *Tetris*. Su simple concepto de jugabilidad se hubiera empañado si las piezas, en lugar de tener formas cúbicas, hubiesen sido de forma esférica. Lo mismo hubiera ocurrido si el conocidísimo protagonista del *PacMan* hubiera tenido forma cuadrada: rompería con la estética global del juego y, posiblemente, ese "comecocos" cuadrado sólo le gustaría a su diseñador. Es muy importante aplicarse personalmente una contundente crítica a la hora de diseñar y construir los gráficos de tu juego. Y si bien deben gustar a uno mismo en primer lugar, las opiniones que otras personas puedan dar son imprescindibles. Al fin y al cabo, en la carrera en busca de una aceptación comercial, son otras personas y el mercado las que juzgan tu trabajo.

Ser metódico y detallista forma parte de esa crítica, pero siempre se debe tener en cuenta el decir "basta, ya está terminado" y cumplir un esquema de desarrollo en un tiempo aceptable. También es importante explotar al máximo las posibilidades de las que se dispone para construir unos gráficos más que notables, siempre dentro de las posibilidades artísticas de cada uno, y sobre todo experimentar con nuevas formas, colores y efectos; este hecho hace más fácil llegar a realizar unos gráficos originales y llenos de personalidad. Está claro que es necesario fijar las miradas en los grandes trabajos ya realizados, pero siempre es mejor diseñar unos gráficos con personalidad, que una buena copia, porque esto ayudará a que el juego resultante sea diferente, original y pueda destacar de entre los demás.

ELEMENTOS GRÁFICOS QUE COMPONEN UN JUEGO

Básicamente todo videojuego se compone de tres grupos de gráficos diferentes:

- **Grupo 1.** Los gráficos que conforman el envoltorio del juego en sí; presentación, gráficos de menús y despedida o fin.
- **Grupo 2.** Los gráficos que forman el conjunto de los protagonistas del juego; jugador, enemigos, objetos (armas, bonus, etc.)
- **Grupo 3.** Los gráficos que componen el entorno del juego; fondos, objetos de decoración, indicadores de pantalla, efectos especiales (explosiones, humo), etc.

Realmente esta clasificación es válida tanto para juegos con visualización en dos dimensiones, como para los de tres dimensiones. Simplemente, cada estilo requiere distintas técnicas a la hora de hacer los gráficos.

8

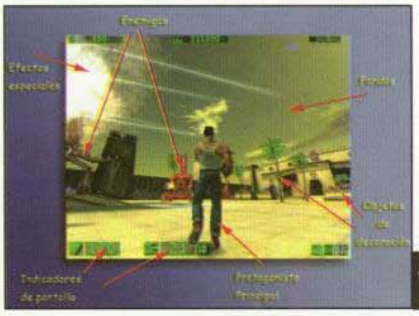
No olvidemos, que unos movimientos suaves forman parte del desarrollo gráfico.

9

Aparte de su original concepción, la serie Worms han triunfado por su representación gráfica.

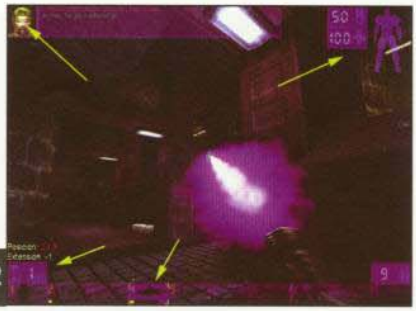
10

Unos efectos especiales realistas ayudan al jugador a introducirse más en la acción.



11

Un ejemplo de los elementos gráficos que componen un juego.

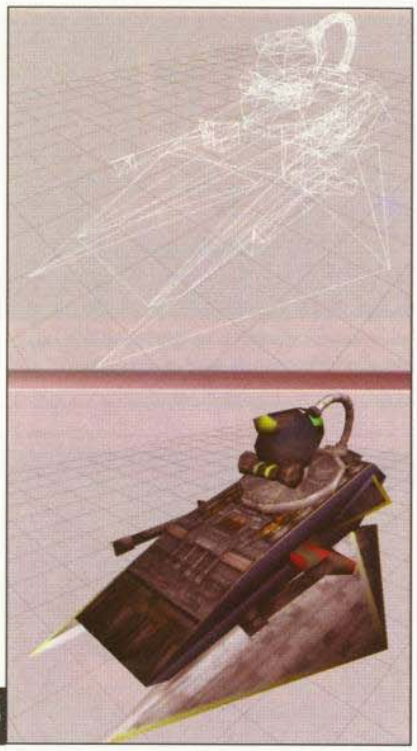


12

Un ejemplo de utilización de bitmaps (sprites) en un juego en 3D.

13

En la imagen podemos ver un juego en 2D con gráficos procedentes de técnicas 3D.



15

El detalle en las texturas puede disimular en gran medida un modelo dibujado con pocos polígonos.

Aunque nuestro videojuego será en 3D, usaremos también gráficos en dos dimensiones que utilizaremos para la presentación, menús, indicadores de pantalla (vida, puntos, energía, nº y tipo de arma), en el editor de niveles y en los ficheros de ayuda. Estos gráficos se pueden dibujar en 2D o bien generar en 3D, pero siempre serán planos, sólo poseen una vista.

En nuestro caso, estos gráficos o *bitmaps* (mapa de bits), y que a partir de ahora llamaremos *sprites*, serán realizados con un programa de dibujo y retoque fotográfico y utilizarán sólo las coordenadas X e Y (horizontal y vertical) de la pantalla.

Los gráficos 3D del juego no son mapas de bits sino polígonos que se dibujan en pantalla en un entorno en tres dimensiones utilizando las coordenadas de los ejes X, Y y Z . Estos objetos poligonales se pueden representar con color o textura para que no sólo reflejen la forma sino también la apariencia. Las texturas no son más que dibujos con forma cuadrada de tamaño siempre múltiplo de 2 (8x8, 16x16, 64x64, 256x256) que cubren los polígonos del objeto. Pero de esto ya hablaremos en próximos números. Y siguiendo con nuestros gráficos en 3D, diremos que en nuestro juego habrá diferentes tipos. Los utilizaremos para representar el escenario donde tendrá lugar el juego, los protagonistas, las piezas del decorado (árboles, piedras, edificios) y también todos los objetos que componen los elementos adicionales, como son los bonus, munición, vida, etc.

NOTA

Unos gráficos cuidados, coherentes con el estilo e historia del juego y unos efectos realistas contribuyen en gran medida a aumentar la jugabilidad.

Blitz 3D posee un motor 3D y 2D que permite mezclar y manipular a la vez la representación en dos y tres dimensiones (siempre dependiendo de las posibilidades de la tarjeta gráfica que se utilice); por lo tanto, no tendremos ningún problema para movernos en un entorno 3D y representar a la vez los *sprites* que sirven de indicadores para la vida, munición, tipo de arma en uso, radar, punto de mira, etc.

Hay que recordar que con los gráficos para nuestro juego, materializamos en gran medida el concepto que esperamos de él cuando lo concebimos en nuestra mente.

NOTA

Por el año 1996 comienzan a aparecer los primeros juegos de acción 3D con enemigos y personajes poligonales. De esa forma se sustituían a los personajes diseñados como *sprites*. La llegada de los procesadores pentium y la tecnología MMX o 3D Now, junto a las librerías DirectX significó una verdadera revolución del videojuego. La transición no fue fácil y hubo juegos totalmente 3D mucho menos jugables que sus versiones anteriores basadas en *sprites*. A la larga, el cambio mejoró la calidad de los juegos, por tanto los modelos 3D poligonales elevaron su realismo a límites insospechados.

En el próximo número...

... hablaremos de las herramientas más comunes que se utilizan para realizar los gráficos de un juego y cuáles elegiremos para llevar a cabo nuestro proyecto.

Conceptos básicos: audio digital

En esta sección aprenderemos el buen arte de dar más espectacularidad y brillantez a nuestro juego, dotándolo de efectos especiales de sonido y música. Debemos conocer, sin embargo, algunos conceptos que ayudarán en gran medida a realizar nuestro cometido con mejor criterio.

¿QUÉ ES AUDIO DIGITAL?

El audio digital se produce cuando una fuente de sonido se introduce en el ordenador a través de la tarjeta de audio y es trasformada en una secuencia de números.

Con esta operación, el ordenador ya está preparado para almacenar el sonido en el disco duro y poder editarlo, transformarlo y aplicarle cualquier efecto. Además, el ámbito digital también proporciona la útil posibilidad de pasar el sonido de un formato a otro según las necesidades.

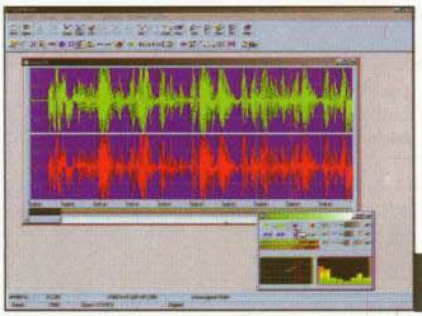
IMPORTANCIA DEL SONIDO EN UN VIDEOJUEGO

En la vida cotidiana, el sonido que nos rodea nos ayuda a distinguir mejor las cosas. Al igual que las imágenes, el sonido queda grabado en el cerebro, quien lo asocia a una idea o imagen. Por ejemplo, no es igual la sensación que nos produce sólo estar viendo el mar que verlo y oírlo a la vez. El sonido refuerza enormemente el significado de las cosas que vemos y nos ayuda a comprenderlas mejor.

Es común utilizar su poder de comunicación en el cine, la televisión y sobre todo en la publicidad. En una película, los efectos de sonido o la música pueden cambiar radicalmente el significado de una secuencia específica. Y por ejemplo, la mayoría de los spots publicitarios que vemos en la televisión los identificamos perfectamente cuando lo oímos por la radio.

En los videojuegos no es distinto el caso. Estoy seguro de que el lector recordará perfectamente cómo comenzaba una partida en el PacMan o, sin ir más lejos, qué sería de una batalla en el Quake III sin ningún tipo de sonido. Es evidente que el audio estimula el entretenimiento y la emoción de jugar, pero para ello debe tener una cierta coherencia. Es fundamental utilizar el sonido adecuado en cada situación; por ejemplo, una música apoteósica o alegre cuando se llegue al final de la aventura, o el sonido de una enorme explosión cuando destruimos la principal base enemiga. A veces ocurre que el uso o no de música enturbia la acción, sobre todo en videojuegos que precisan de toda la atención al sonido del entorno, como, por ejemplo, el rugir de una criatura detrás de una puerta.

También debemos tener en cuenta la cantidad de sonido que utilicemos en distintas partes del juego y el volumen de cada uno de ellos. Generalmente se utilizan los sonidos propios de cada acción, ya sean de protagonistas como de los objetos del entorno, como puertas, interruptores, etc. Si estamos en medio de una batalla y disparamos con un arma, es normal que suenen el disparo y el impacto, pero no es necesario que se oi-



1

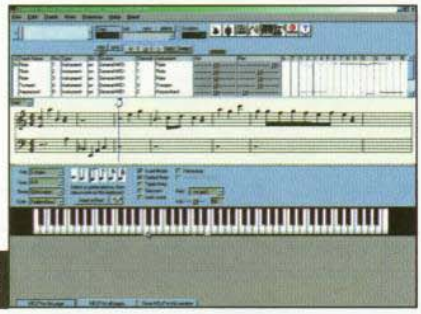
Los editores de audio digital te permiten editar los sonidos y aplicarles multitud de efectos.

2

Poder importar y exportar en distintos formatos y convertir a diferentes calidades es fundamental.

TRUCO

Asignar un sonido a una acción determinada dará más profesionalidad al juego. Por ejemplo, cuando activamos un interruptor, chocamos con cualquier parte del decorado o elegimos una opción en el menú.

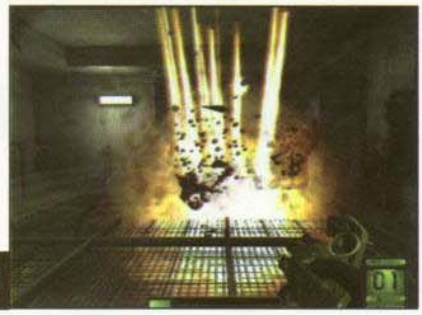


3

Los editores de audio digital te permiten editar los sonidos y aplicarles multitud de efectos para luego ser aplicados a nuestros juegos.

4

Anvil Studio es un secuenciador MIDI Shareware con todas las posibilidades de programas más profesionales.



5

Muchos secuenciadores permiten mezclar pistas de audio y MIDI simultáneamente con gran maestría.

6

Hoy en día es fácil encontrar un buen programa para hacer tu propia música sin necesidad de instrumentos externos.

ga también la trayectoria del proyectil, sobre todo si en la misma escena hay diez armas disparándose a la vez. Esto saturaría la acción si además hubiera una música de fondo.

Para crear cierta atmósfera es muy común utilizar sonidos de ambiente. Proporcionan más realismo al juego ayudando al jugador a involucrarse más en la partida.

Miremos la importancia del audio, por ejemplo, en el avance continuo que existe en cuestión de sonoridad para los ordenadores personales: se ha pasado de unos simples pitidos del altavoz interno hasta estaciones *surround* de múltiples altavoces y sistemas de *dolby digital* propios del cine. Hay que tener claro, pues, que los efectos especiales de audio y la música juegan un papel muy importante en el desarrollo de un videojuego y pueden ser vitales para el éxito o no de un proyecto.

ELEMENTOS SONOROS QUE COMPONEN UN VIDEOJUEGO

En todo videojuego debe de existir algún tipo de efecto de sonido asociado a una acción determinada. Estos sonidos producen una información adicional que el jugador siempre agradece, ayudando a mejorar el estímulo y a definir mejor la acción. Al igual que cambiamos un color cuando activamos una opción en el menú, con un leve sonido de activación podemos reforzar aún más esa comunicación. Un claro ejemplo de un efecto de audio adecuado es el aplicado a las pisadas del protagonista, cuando suenan de distinta forma dependiendo del suelo que pisa o, por ejemplo, el sonido de alivio al salir del agua y poder respirar; por un lado producen más realismo a la acción y por otro indica al jugador que se ha alcanzado la superficie.

Otro elemento, que cada día se está implantando más en los juegos, es la voz, bien de fondo, narrativa o diálogo de personajes. Se utiliza mucho para ambientar aún más, proporcionar ayuda, explicar ciertas acciones o misiones durante el juego, y en tutoriales de aprendizaje.

No podemos olvidar la música como elemento de estímulo, ambientación e identidad. Recordemos la importancia que tuvo en las películas mudas y la acción añadida que puede llegar a proporcionar en ciertas situaciones. Es tan importante, que algunas bandas sonoras de juegos son verdaderas obras sinfónicas y, cada vez más, se pueden adquirir por separado en un CD de audio, al igual que ocurre con el cine.

Básicamente hay una música distinta en cada parte del juego: en la presentación, en el menú, en cada cambio de ambiente o nivel, en el final de la partida y en el final del juego. Actualmente, se utiliza la música continuada, que no es más que varias melodías enlazadas durante la partida que cambian según la acción.

Es muy importante que el audio de un videojuego guarde coherencia con las imágenes y acciones. Que la música esté acorde a la temática de la historia y sobre todo, buscar la calidad.

NOTA

Los secuenciadores MIDI permiten realizar temas musicales mezclando varias pistas y aprovechar todas las posibilidades de la tarjeta de sonido.

En el próximo número...

... aprenderemos lo necesario sobre los formatos de audio más utilizados en un videojuego y cuál de ellos utilizaremos.

Programar con Blitz 3D

Hoy en día, gracias al avance informático, la fuerte implantación de las consolas de juegos en el mercado y por supuesto a Internet, los videojuegos se han convertido en la alternativa mayoritaria en el mundo del ocio para todas las edades.

Todas estas premisas han contribuido a que el usuario de a pie tenga acceso a infinidad de herramientas y lenguajes de programación para la realización de aplicaciones multimedia y desarrollo de videojuegos. Sin embargo, muchas de estas herramientas aportan una excesiva dificultad de uso y un coste elevado. Por este motivo, y aprovechando este tipo de demanda, apareció *BlitzBasic*, una herramienta de programación fácil de utilizar y a la vez potente.

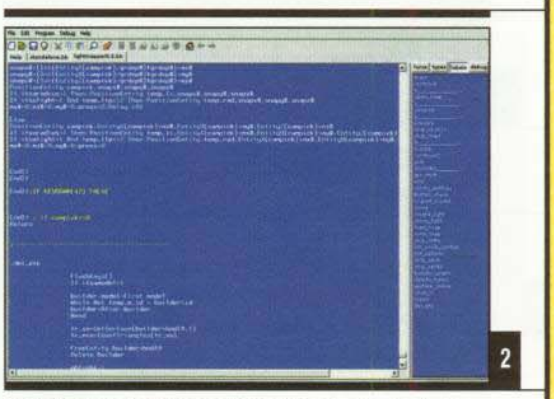
NOTA

***Blitz 3D* es la herramienta que utilizaremos para aprender a programar un videojuego, en el CD-ROM se encuentra una demo totalmente operativa con la que seguir el curso y programar los ejemplos.**

ORIGEN DEL BLITZ BASIC

Blitz Basic nace en EEUU de la mano de Mark Sibly, quien pretendía llevar al programador principiante el poder de las *DirectX* de *Microsoft* con la sintaxis fácil y flexible del lenguaje *BASIC*. Esta filosofía hizo que poco a poco esta herramienta alcanzase una acogida cada vez mayor entre los usuarios deseosos de entrar en el mundo del desarrollo de videojuegos.

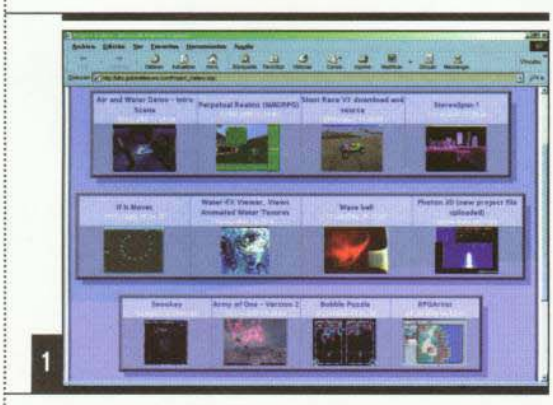
Además de un bajo coste (alrededor de 60 Euros) y su aceptable potencia, cumple otros requisitos plausibles como son: no tener que pagar licencia alguna para vender aplicaciones realizadas con él; perfecta curva de aprendizaje, ideal para programadores noveles; y el atractivo más que apreciable para desarrolladores experimentados. El lenguaje, en el más estilo *BASIC* con mezclas de otros lenguajes como C o Pascal, comienza su andanada como herramienta de programación de juegos en 2D, como lo fueron en su día AMOS para Amiga y Atari o más recientemente DIV para PC. Su estructura de diseño más sobresaliente es su modularidad, la cual permite dividir el programa principal en diferentes subprogramas por separado. Este sistema ayuda al depurado del código y a la ampliación de las posibilidades de una aplicación. Asimismo, permite una sintaxis muy flexible que facilita la realización de códigos optimizados y con personalidad.



2

Tanto BlitzBasic 2D como Blitz 3D incluyen un estupendo editor.

Sus características gráficas 2D son sobresalientes: comandos para dibujar bloques, rectángulos, imágenes, elipses y líneas. Incluye la utilización de los formatos de imagen más populares (.bmp,



1

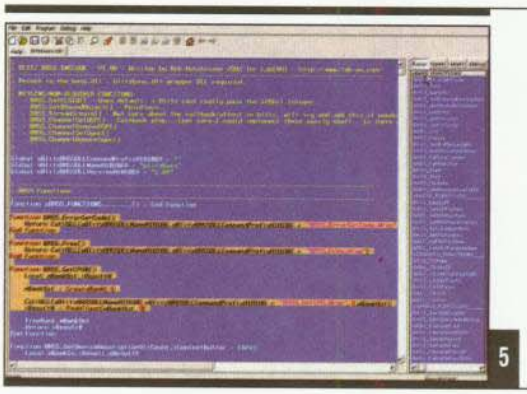
En la página de Blitz Basic (www.blitzbasic.com) existen todo tipo de videojuegos desarrollados con esta herramienta.

DEFINICIÓN

El sistema de doble búfer (double buffering) consiste en dibujar el gráfico en una zona de la memoria (búfer) antes de representarla en la pantalla. Cuando ya está dibujada, se vuelca esa zona de memoria a la zona de pantalla, y así sucesivamente. De esta forma se evita el parpadeo del gráfico en cada refresco del monitor.

3

La gran comunidad de usuarios que Blitz Basic tiene en Internet posibilita el intercambio de herramientas para el desarrollo con este lenguaje de programación.



5

Para ampliar las posibilidades del lenguaje, Blitz Basic incluye la posibilidad de importar librerías externas de funciones e incluso librerías de enlace dinámico (DLL).

.jpg, etc.). También posee un sistema de representación en pantalla muy suave gracias al uso del *doble búfer*. Igualmente, admite animaciones de gráficos frame a frame (cuadro) y un sistema de colisión por píxel (punto).

Más características que ofrece *Blitz Basic* son la manipulación de archivos y recogida de datos de entrada desde diferentes conexiones como: teclado, ratón y multitud de dispositivos para juegos, además de funciones externas FTP para multijugador.

No podemos dejar atrás las numerosas herramientas de que dispone, la mayoría desarrollada por los propios usuarios, como editores de terrenos, mapas o programas de dibujo.

Blitz Basic tiene una comunidad envidiable en Internet y está presente en multitud de páginas web especializadas del sector en numerosos idiomas. También cabe destacar que hay gran cantidad de juegos, de diferentes géneros, desarrollados con *Blitz Basic* que han sido comercializados.

Atendiendo a la demanda y avance del sector, el equipo de *Blitz Basic* se vio obligado a volcar todos sus esfuerzos en desarrollar una nueva versión con posibilidades 3D en un producto por separado: *Blitz 3D*.

CARACTERÍSTICAS DE BLITZ 3D

Este lenguaje de programación nace con la idea de llevar el entorno de programación del *Blitz Basic 2D* original al campo de las tres dimensiones sin dejar en ningún momento la filosofía de sencillez y potencia que siempre ha caracterizado a esta herramienta de desarrollo multimedia.

En esta ocasión, se han conservado todas las funciones 2D y se le ha añadido un grupo, bastante generoso, de instrucciones para el manejo de las 3D. Al igual que su antecesor, el corazón del *Blitz 3D* es un entorno de desarrollo integrado (IDE) donde se puede escribir, testear y ejecutar los programas. Dispone de una extensa gama de posibilidades, entre las que se encuentra un gran depurador de código y un visualizador de variables que ayuda al programador, en cualquier momento, a obtener información del proceso de ejecución y poder corregir rápidamente errores en el código.

Muchas son las características que esta aplicación nos ofrece. Empezando por el compilador que, gracias a que produce 100 por 100 código máquina, se obtiene un rendimiento de velocidad más que satisfactorio en los ejecutables. También, los programas podrán correr a pantalla completa o en una ventana de escritorio configurable.

Disponemos de un generoso abanico de tipos de datos numéricos: *boleanos, enteros, byte, coma flotante.*

4A 4B

El buen resultado que ofrece Blitz 3D ha facilitado la comercialización de muchos juegos programados con esta herramienta de desarrollo. Muchos juegos desarrollados por amateurs se venden a través de distintas páginas web.

6

Juegos en 2D de gran calidad también son posibles gracias a Blitz 3D.

Así como la posibilidad de utilizar *constantes*, *funciones* y *arrays*. Además, se pueden utilizar *tipos* para crear estructuras que se emplearán para definir las características de los elementos que intervienen en el juego. Igualmente, se pueden crear funciones definidas por el usuario e incluir subprogramas en el programa principal.

Aunque algunos de estos conceptos puedan sonar a chino, no os preocupéis: a medida que avancemos en la obra no sólo aprenderéis qué significan, sino a utilizarlos adecuadamente.

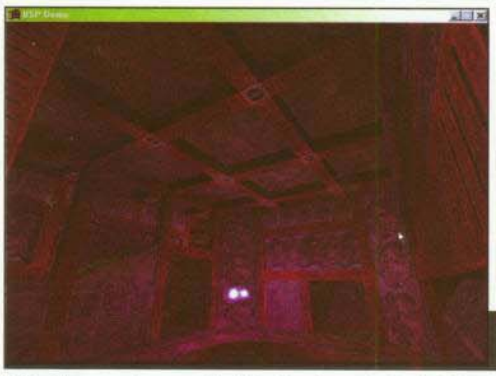
Pero *Blitz 3D* se diseñó para el desarrollo multimedia y sobre todo para la creación de videojuegos, así que también ha sido ampliado en los formatos de gráficos, audio y vídeo que es capaz de manejar. En el ámbito de las 3D, al ya tradicional formato de objetos de *DirectX (XoF o .X)* y 3D Studio *(.3DS)*, se le ha añadido la posibilidad de utilizar el formato de animación de objetos .MD2 (animación por deformación de malla.) Y una fantástica novedad, la posibilidad de manipular sistemas o niveles *BSP*, ampliamente extendido hoy día para el mapeado de juegos –principalmente de tipo FPS (Shooter en primera persona)–, como son los mapeados del *Quake II, III* y *Half Life*.

En el apartado de audio y vídeo encontramos el manejo de los formatos más utilizados como: *.WAV, .MP3, .X3M* o *.MIDI, .MPEG, .AVI* y la posible aplicación de efectos de posicionamiento 3D del sonido.

Antes de entrar de lleno en las posibilidades del motor 3D que Blitz 3D posee, hay que comentar algunas características que lo hacen sobresalir más si cabe, como son: la inclusión de funciones FTP para multijugador en red de área local (*LAN*) e Internet, perfecto funcionamiento en Windows XP y total compatibilidad con programas hechos en *Blitz Basic 2D*. No olvidemos la capacidad de crear y modificar ficheros, así como un sistema de lectura aleatoria y secuencial de datos binarios o de textos.

Y lo mejor de todo, el usuario está absolutamente libre de pagar licencias o royalties a la hora de vender su trabajo, al igual que ocurría con el antiguo *Blitz Basic 2D*.

Pero si *Blitz 3D* se distingue de la mayoría de lenguajes es por la potencia de su motor gráfico. Es capaz de representar, transformar y mover objetos en 3D y 2D, detectar colisiones entre ellos y además aplicarles texturas y multitud de efectos de visualización. Puede manejar y transformar terrenos; crear y manipular luces en tiempo real; y tiene la posibilidad de añadir efectos especiales como niebla y transparencias, entre otras.



7

Blitz 3D posee la capacidad de visualizar niveles BSP. Es posible interactuar con niveles de la saga Quake.

NOTA

Boleanos, enteros, byte, coma flotante, constantes, funciones, arrays, mapeados, mallas, etc. son términos que serán estudiados y explicados a lo largo del curso. El lector no debe preocuparse si ahora no le queda claro algún concepto ya que se abordarán en futuras entregas.

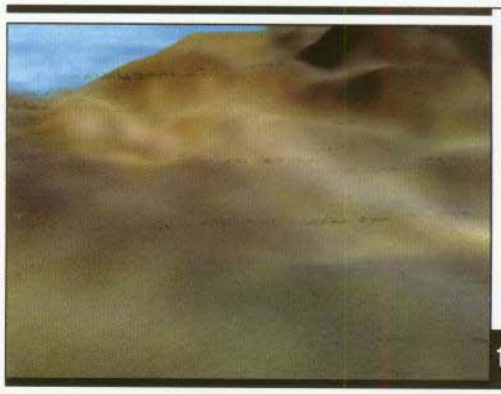
8

El formato .MD2 es muy utilizado hoy día para animar a los personajes.

9

El motor 3D del Blitz 3D permite representar grandes superficies exteriores sin ningún problema.

Blitz 3D utiliza un método revolucionario y flexible para la administración de todos los objetos 3D. Se trata de la utilización de un sistema de "entidades", consistente en definir a todos los elementos 3D como entidades. Este inteligente sistema, facilita un método de asignaciones muy útil, pudiéndose controlar el estado, el movimiento o las colisiones de luces, cámaras o superficies como si fueran objetos.



10

En la imagen superior vemos un bonito ejemplo de multi-texturizado de terrenos.

También está preparado para renderizar en tiempo real tanto interiores como grandes superficies y utiliza las posibilidades de transformación e iluminación por hadware (*T&L*) aprovechando la aceleración gráfica de las tarjetas. Posee la característica de emplear técnicas para optimizar el rendimiento como la utilización del nivel de detalle o *LOD*, el cual básicamente consiste en reducir los polígonos de los objetos a medida que se alejan de la cámara.

NOTA

Un buen ejemplo del sistema de entidades que utiliza *Blitz 3D* sería poder detectar, prácticamente con una sola instrucción, la colisión entre la entidad "cámara" y la entidad "objeto pared.

DEFINICIÓN

La técnica Mip Mapping, consiste en dividir una textura a la mitad y así sucesivamente. De esta forma podemos aplicar distintos tamaños de texturas al objeto, dependiendo de la distancia que esté de la cámara.. Es un procedimiento muy usado para aumentar el rendimiento de un juego.

El manejo de las texturas es sobresaliente en esta herramienta. Posibilita técnicas de *mip mapping*, transparencias, técnicas de *multi-texturizado* de hasta 8 texturas a la vez y soporte para texturas animadas. De nuevo insistimos en que todas estas técnicas serán explicadas para los más novatos en programación.

Además, permite colorear los vértices de un objeto, admite *environment mapping* (reflejos en el objeto del entorno) y otros efectos de reflexión.

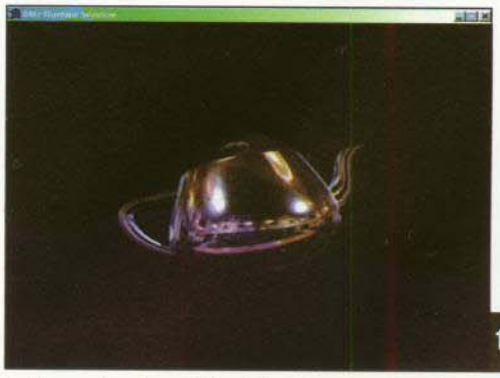
Blitz 3D puede crear y manipular distintos tipos de luces: ambientales, focos de luz y direccionales. Acepta establecer varias vistas de cámara al mismo tiempo y posee un sistema de colisiones entre objetos de gran velocidad de respuesta. Puede crear pinceles, *meshes* (objeto con varios polígonos) y superficies, y aplicarles una gran diversidad de potentes operaciones, como el cambio de apariencia y forma de los objetos en tiempo real.

Además admite otras características muy interesantes como son: el control suave en las transiciones entre animaciones y un sistema avanzado de interpretación de mapas de alturas para la generación de terrenos muy fácil de usar, así como la posibilidad de aplicarles varias texturas a la vez.

En definitiva, *Blitz 3D* es una herramienta potente, fácil de aprender y con un bajo coste (145 euros), ideal para principiantes y profesionales que pretendan realizar una aplicación multimedia o videojuego de cualquier género.

En el próximo número...

... entraremos de lleno en su aprendizaje, viendo en primer lugar las características y funcionamiento del editor de programas.



11

Arriba un ejemplo práctico del efecto que genera el mapeado de entorno (environment mapping).

Consejos para mejorar un videojuego

En este primer tutorial de la obra, hemos querido comenzar dando algunos consejos sobre cuestiones que muchas veces se olvidan o se consideran en segundo lugar y que, sin embargo, significan la posible aceptación o no de un proyecto.

La creación de un videojuego, aunque resulta divertida y excitante, implica un gran esfuerzo, ya que hay que tener en cuenta muchos factores: criterio artístico, tecnología del software e incluso mucha psicología. Todos estos apartados deben ir de la mano, en coherencia con la idea global. Resulta incorrecto mezclar conceptos en un mismo proyecto como la utilización de música relajante en juegos de combate o utilizar armas futuristas en una aventura medieval. Incluso hasta el diseño de letra que utilicemos en los textos es importante.

EL CONCEPTO DE JUGABILIDAD

Es fundamental cuidar la calidad total del proyecto. Un juego no será bueno por llevar unos gráficos increíbles, también influye la calidad de la música o los efectos sonoros y lo que es básico, la jugabilidad.

NOTA

Lo que más agradece un jugador es precisamente poder jugar y divertirse con ello.
Así que el fin que debemos darle a nuestro videojuego es que tenga jugabilidad sobre todo lo demás.

¿Qué interés tiene un juego con cientos de pantallas, con gráficos perfectos o efectos especiales fantásticos si es imposible culminar las primeras pantallas? Un claro ejemplo del poder que ejerce la jugabilidad en la calidad total lo encontramos en videojuegos tan sencillos estéticamente como el *Tetris*, al que se sigue jugando 15 años después de su nacimiento. Para que un juego tenga más o menos jugabilidad hay que estudiar bien el diseño de la acción, añadir una dificultad progresiva o dar incentivos al jugador a medida que resuelve puzzles o pasa niveles. Evidentemente, el realismo que se pueda conseguir en representar la acción influye notablemente en la jugabilidad; por ejemplo, las sombras que se proyectan en ciertos juegos ayudan al jugador a descubrir enemigos tras una esquina, o en el caso de un vuelo sobre un terreno montañoso, a calcular la altura de la nave.

EL MENÚ PRINCIPAL

Un videojuego, además de la acción propiamente dicha, se compone de otros elementos imprescindibles para su buen uso, como es el menú. Esta pieza importantísima enseña al usuario todas las posibilidades que ofrece el producto. Pero antes debemos conducir al jugador al mundo de fantasía que hemos creado. Para ello se suele utilizar una buena presentación de la historia por medio de animación o dibujos antes de entrar en el menú.

En el menú comienza realmente el juego y es la primera impresión que el jugador tendrá de él. Por ello, debe poseer la misma estética, un sencillo y claro interfaz de uso y sobre todo un buen abanico de opciones. Debemos dar al usuario la facilidad de cambiar todos los aspectos posibles del juego. Lo más común es la posibilidad de cambiar los volúmenes de audio, la calidad gráfica como resolución de pantalla y brillo o la configuración de los controles.

Por ejemplo, podemos definir más ampliamente la calidad gráfica que tendrá el juego, dependiendo de la tarjeta gráfica o de la potencia del ordenador, con factores

En todo menú debe existir un apartado de opciones para configurar el juego a medida del jugador.

2

Configurar los máximos aspectos posibles del protagonista principal ayuda a que el jugador se identifique más con su personaje.

como el tamaño de las texturas, la profundidad de color, la activación o no de luces y sombras dinámicas y un buen número de cosas más.
Permitir al usuario cambiar los controles del teclado o elegir un dispositivo de juegos externo se hace casi imprescindible.

Sin embargo, hay muchas más opciones que se pueden incluir y que nos ayudarán a mejorar el producto; una de ellas es que el jugador pueda identificarse más con su personaje permitiéndole modificar el aspecto del protagonista o darle un nombre.

Para reforzar la importancia del jugador se puede incluir una tabla de records, si en nuestro juego se acumulan puntos, o halagar al jugador durante el juego con frases como "estupenda decisión" o "fantástico, eres el mejor..." y cosas así. Algún tipo de tutorial dentro del menú hará que nuestro juego gane puntos y gane en calidad.

3

Es fundamental que el juego vaya acompañado de una ayuda en formato de texto o HTML.

OTROS AÑADIDOS

Otros aspectos a tener en cuenta son las herramientas que rodean al producto. El desarrollo de un videojuego no termina con la finalización del juego en sí, hay que proporcionar al usuario una serie de facilidades y utilidades que le ayuden a utilizarlo lo mejor posible.

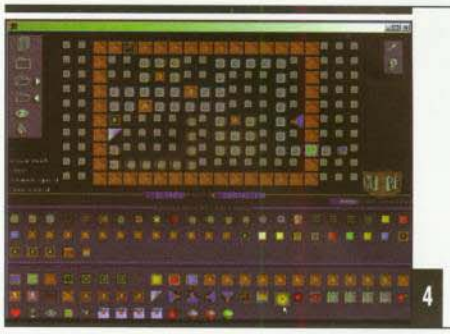
Es primordial incluir una información adicional como ayudas de instalación, manuales de uso, soporte técnico, descripción del juego o requisitos para poder jugar. Los manuales de uso y cualquier otra información, pueden ir en distintos formatos, desde formatos de texto hasta HTML.

Un punto que cada vez está más extendido es la inclusión de un *editor de niveles*, bien externo o, dentro del juego, en el menú principal. Esto nos ayudará a alargar la vida de nuestro juego y conseguir un mayor interés por parte del usuario.

Otra cuestión fundamental es proporcionar al videojuego un sistema de instalación guiada, aunque siempre se agradece una instalación automática con autoarranque una vez introducido el CD.

Por último hay que tener cuidado con los *copyrights* y licencias si vuestro proyecto pretende tener una salida comercial, porque existen leyes muy rígidas que regulan este tipo de asuntos. Por ejemplo, hay programas y ciertas librerías de programación que necesitan el pago de una licencia. Tened cuidado con el plagio en las músicas, el origen de vuestros sonidos y gráficos o incluso las fuentes de letras y logotipos.

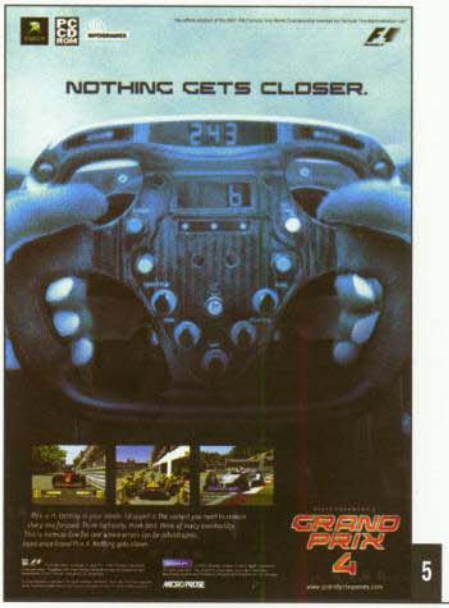
Reservaremos esta sección de la obra para aquellos aspectos adicionales necesarios para un desarrollo más profesional. Aprenderemos aquí a conocer herramientas extendidas en el mundo del videojuego como editores de



4

Incluir un editor de niveles ayudará a alargar la vida del videojuego, además de proporcionar un buen reclamo comercial.

niveles y audio, secuenciadores, cajas de ritmos, editores de vídeo y efectos para nuestras presentaciones. También terminaremos aquí el desarrollo de nuestro juego, con la realización del tutorial y el sistema de instalación. Se aprenderán trucos de programación, rutinas especiales y consejos muy útiles.



5

Al final de todo el proceso el éxito de un juego puede depender también de una buena campaña publicitaria.

En el próximo número...

... comenzaremos explicando uno de los editores de niveles BSP más utilizados en la actualidad, el *Q3RADIANT* de Id Software.

Evolución técnica del videojuego

Los juegos han existido a lo largo de la Historia del hombre. Los más comunes han sido tradicionalmente los de mesa y de salón. El ajedrez, las cartas o el billar han acompañado nuestro ocio durante siglos. Pero no fue hasta la llegada de la electricidad cuando los juegos cobraron otra dimensión.

La combinación de corriente eléctrica y piezas mecánicas contribuyó al desarrollo de juegos de arcade electromecánicos entre los años cincuenta y sesenta, creándose circuitos de coches de carreras, tiro al blanco móviles o la máquina de diversión mecánica por excelencia, el pin-ball. Sin embargo, estos aparatos eran de grandes dimensiones y muy complejos de mantener, por lo que estaban relegados sólo a sitios públicos y difícilmente se podían tener en casa.

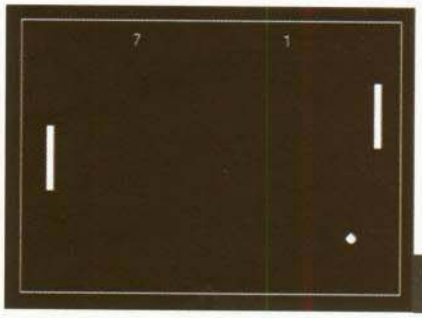
EL PRIMER VIDEOJUEGO

No será hasta la llegada de la electrónica y la informática cuando nazca el *videojuego*. En el año 1973, como si de una plaga se tratará, apareció -en infinidad de locales públicos como bares, aeropuertos, salones y heladerías- el primer arcade totalmente electrónico, el *Pong*. A pesar de su enorme sencillez, causó un gran revuelo en la época debido a que carecía totalmente de partes mecánicas móviles; todo el juego estaba implementado en un microprocesador y se desarrollaba en la pantalla de un televisor. Consistía básicamente en hacer rebotar una pelota cuadrada con una paleta en forma de rectángulo vertical (todos del mismo color), que cada jugador poseía, y evitar que se saliese por los lados. Tenía similitud con un partido de tenis o ping-pong y se podría considerar como el primer arcade simulador deportivo de la historia. Pronto adquirió distintas variantes como el frontón o la doble paleta por jugador. Debido a su carencia total de violencia, su temática deportiva y de habilidad, y la posibilidad de ser introducido en una caja pequeña contribuyó a que invadiera, poco a poco, los hogares de medio mundo.

LOS MATAMARCIANOS Y EL COMECOCOS

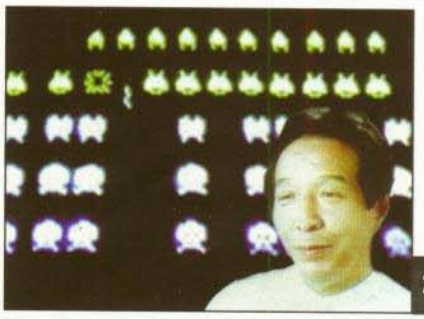
Sin embargo, la popularidad definitiva del videojuego no marcó historia hasta la llegada del primer matamarcianos, creado por el señor Nishikado en Junio de 1978, el *Space Invaders*. Básicamente consistía en que el jugador debía defender su base terrestre de una horda invasora de naves del espacio. Para ello disponía de un tanque o cañón que movía horizontalmente en la parte inferior de la pantalla. A medida que el tiempo pasaba los enemigos avanzaban a mayor velocidad. El juego poseía efectos especiales de sonido muy simples en forma de disparos y explosiones y varios colores en pantalla. Fue una verdadera revolución y quizás sea el videojuego que ha tenido más adaptaciones para recreativas, ordenadores personales y consolas.

Allá por el año 77, un autodidacta japonés, Tori Iwatami, que no era informático y ni siquiera tenía idea alguna sobre arte, dibujaba en una servilleta



1

El Pong, en el año 1973, fue el primer videojuego y simulador deportivo de la historia. Se difundió rápidamente por los hogares de todo el mundo.



2

La idea original de Nishikado era crear blancos móviles por la pantalla para que el jugador les disparase.

3

Una imagen de la versión original de recreativas del TETRIS, quizás el videojuego más popular y con más versiones de toda la historia.

algo parecido a círculos con ojos y una enorme boca a su lado. Se podían distinguir también algunos trazos con forma de fantasmas, al estilo de cuentos para niños. Eran los diseños de uno de los videojuegos más simples, adictivos y controvertidos jamás creado, el *PacMan* o *"Comecocos"*.

Fue convertido en videojuego después de cerca de año y medio de trabajo con la ayuda de otras cuatro personas. Constaba de un laberinto lleno de puntos en el que se hallaban el jugador (comecocos) y cuatro fantasmas que lo perseguían. El nivel terminaba cuando todos los puntos eran devorados. A medida que los niveles aumentaban, la velocidad y el laberinto cambiaban. Este simple concepto iba adornado por una música y efectos de sonidos tan característicos que todavía hoy día se puede distinguir un *PacMan* sólo con oírlo. Además incorporó por primera vez secuencias animadas entre paso de niveles, permitiendo así que el jugador tomara un respiro mientras avanzaba la dificultad de la partida.

El *PacMan* todavía se sigue adaptando a multitud de versiones e incluso hay, como ya hemos visto en la zona de gráficos, "Comecocos" en 3D, y es tanta su jugabilidad que el sobrenombre de "Comecocos" tuvo su influencia por las críticas que recibían sus jugadores por gastar horas y horas delante de la pantalla.

Pero hay un videojuego que atrapó a jugadores de todas las edades y ambos sexos y que ha estado presente en todas las formas conocidas de máquinas de juegos; nos referimos sin duda al *Tetris*. Seguro que no hay una sola persona en el mundo que no haya visto alguna vez este videojuego en acción. Consistía en encajar diferentes formas geométricas entre sí para terminar una línea horizontal, la cual desaparecía. Las piezas, antes de caer, podían ser giradas y desplazadas horizontalmente por el jugador y el juego terminaba cuando las piezas mal colocadas y acumuladas unas sobre otras agotaban el espacio de movimiento para nuevas piezas.

Además fue el primer videojuego de desarrollo soviético y fue diseñado por Alexey Pajitnov y Dmitry Paulovsky y pasado al IBM PC por Vadim Gerasimov entre 1985 y 1986. Realmente la inspiración de Pajitnov para diseñar el *Tetris* partió de un juego llamado *Genetic Engineering* en el que el jugador movía 4 piezas con forma cúbica que podía ensamblar entre sí. Curioso, ¿verdad?

Para terminar, no podemos dejar atrás la versión más conocida y afamada del viejo juego *The Wall* (el muro): nos referimos al *Arkanoid*. Consistía en derribar una pared de ladrillos por medio de una bola que iba rebotando continuamente entre los ladrillos, los extremos horizontales de la pantalla y la paleta que controlaba el jugador. A medida que se avanzaban niveles, la pared cambiaba de forma y a veces los ladrillos al romperse soltaban ciertas piezas que caían y que al chocar con el jugador activaban diferentes características en el juego, como por ejemplo más velocidad de la bola, una paleta más ancha o simplemente el jugador sumaba puntos.

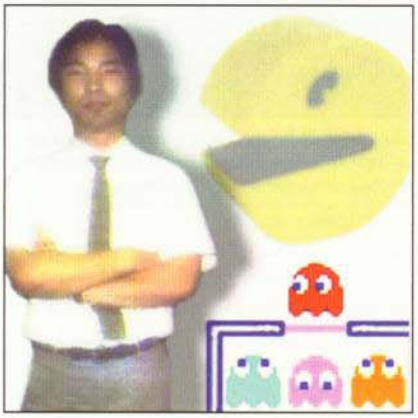
No se puede terminar sin mencionar otras viejas glorias de recreativas que han marcado el futuro de los videojuegos como son: *Defender, Galaxian, Asteroids, Phoenix, Lunar Lander, penetrator* y muchos más.

LA BIOGRAFÍA...

TORU IWATANI

Diseñador del **PacMan**

Nació en Tokio el 25 de enero de 1955.
A la edad de 22 años en 1977 entró a trabajar en la compañía de videojuegos Namco Limited con algunos dibujos de su juego en el bolsillo. Era autodidacta y no tenía ninguna formación en informática, en arte o en diseño gráfico. Se ayudó para completar el *PACMAN* de un ingeniero en hadware, un músico, algunos diseñadores gráficos y un programador.
Trabajaron codo con codo durante un año y cinco meses y fue el algoritmo que movía a los cuatro fantasmas lo más duro de implementar ya que Iwatani quería proporcionar características y movimientos distintos a cada uno de ellos, dar a cada fantasma un comportamiento propio dependiendo del movimiento y posición del jugador y de los otros fantasmas.
Nacía así la primera implementación de inteligencia artificial compleja en un videojuego.
Después de terminar el *PACMAN* diseñó *Libble Rabble*, pero no obtuvo el mismo éxito en recreativas.



En el próximo número...

... hablaremos de los primeros juegos para ordenadores domésticos: el primer paso de los videojuegos hacia una forma inevitable de ocio en los hogares.

Cuestionario Videojuegos 1

Preguntas

1. ¿ Qué característica principal tiene el nuevo compilador del *Dark Basic Profesional*?
2. ¿ Qué son los *vertex shaders* y qué posibilidades nos permite realizar?
3. ¿Qué es un diagrama de GANTT?
4. ¿Cuáles son las tareas que conforman el diseño de un videojuego?
5. ¿ Qué es lo más importante a la hora de realizar los gráficos para un videojuego?
6. ¿Cuáles son los elementos gráficos que componen un videojuego?
7. ¿Qué se entiende por *audio digital*?
8. ¿Cuáles son los elementos sonoros que forman parte de un videojuego?
9. ¿Qué elemento nunca debe faltar en el menú de un videojuego?
10. **¿Cuál fue el primer videojuego de la historia y qué lo distinguía de los demás juegos electromecánicos?**

A partir del próximo número, esta sección se compondrá de preguntas y las respuestas del número anterior.

Contenido CD-ROM 1

▶ AUDIO

En este apartado os ofrecemos las herramientas necesarias para crear y editar audio en todos los formatos: .MIDI, .WAV, .MP3, etc... Para que a vuestro juego no le falte una buena ambientación sonora y musical.

■ Anvil Studio 2002.05.03

Podremos reproducir, grabar, componer y editar pistas de música en formato .MIDI o .WAV.

■ Cool Edit Pro 2.0

Uno de los editores de audio más utilizado por la gran profesionalidad de sus resultados.

■ CuBase VST 32

Con esta nueva versión es posible manipular tanto MIDI como pistas de audio.

■ Goldwave 4.6

Podremos manejar el formato .MP3 y añadir efectos especiales a nuestro juego.

■ Modplug Tracker 1.16.0203

Tracker para realizar secuencias de muestras.

■ Sound Forge 6.0

Otro potente editor de audio, para manipular y aplicar efectos de sonido.

DISEÑO 2D

El diseño 2D es muy necesario para crear las primeras muestras del juego, los primeros bocetos y storyboards. Para ello es necesario contar con unas adecuadas herramientas de dibujo y retoque; esto nos servirá como punto de partida para darle la primera forma a los gráficos de nuestro juego.

■ Deep Paint 1.1

Potente aplicación que convierte imágenes e ilustraciones en lo que nosotros deseemos.

■ Deluxe Paint 2

Un sencillo pero a la vez útil editor de imágenes que hará las delicias de los más nostálgicos.

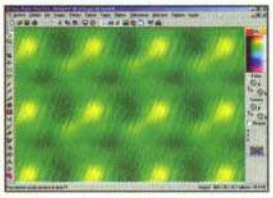
■ Font Creator 3.0

Editor de fuentes TrueType, para crear y editar nuestros propios ficheros TTF.

■ GIMP 1.2.0

Alternativa gratuita al Photoshop, con excelentes prestaciones.

■ PaintShopPro 7

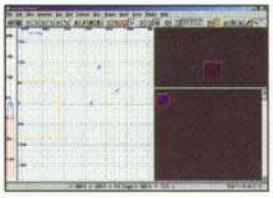


Podremos crear, editar y retocar imágenes de un modo profesional y potente.

DISEÑO 3D

Una vez hayamos diseñado nuestros personajes, tendremos que movernos en un entorno 3D para comenzar con el juego propiamente dicho.

■ Q3 Radiant 199



Editor de mapas para crear y dibujar los nuestros propios.

■ Bryce 5

Estándar para la creación de mundos artificiales con fantásticas posibilidades.

■ Character FX 1.3.1

Manipulador de cada fotograma de la animación, para tener un control absoluto sobre ésta.

■ Deep Paint 3d 2.0

Herramienta profesional para el texturizado de objetos 3D.

■ Lithunwrap 1.2

Creación de plantillas para el texturizado y optimización de objetos 3D.

■ Now 3d 1.20.83

Aplicación tipo RayTracing para generar imágenes en 3D de manera muy fácil.

■ Milkshape 3d 1.5.10

Modelador 3D en baja poligonización y fácil de usar.

PROGRAMACIÓN

Aquí os presentamos una serie de herramientas que os servirán para la parte de programación e instalación del juego.

■ Blitz 2d 1.60

Un clásico en la programación de videojuegos, que tiene una gran cantidad de seguidores.

■ Blitz 3d 1.66

Nuestra herramienta principal, con la que aprenderemos a programar juegos en 3D.

■ GCC

Colección de compiladores de libre distribución.

■ Inno Setup 3.0.2

Generador de Setup, esto es, constructor de paquetes de instalación de software tan potente como versátil.

■ Watcom 1.0

Compilador de C y C++ potente y de gran renombre.

■ Dev-C++ 4.0

Compilador con todas las herramientas necesarias para escribir, compilar, depurar y ejecutar programas en C.

EMULADORES Y JUEGOS

Este mes os animamos a divertiros con dos clásicos: el Spectrum y el Commodore 64, en dos emuladores muy sencillos de usar que incluyen además juegos para los nostálgicos.

■ Emulador de Spectrum z80 1.03

Incluye 21 juegos en versión emulada y dos juegos clásicos adaptados al PC: el Manic Miner y el Pacworld.

■ Emulador de Commodore 64 2.0

Incluye los juegos Commando, MrDo y Pacman.

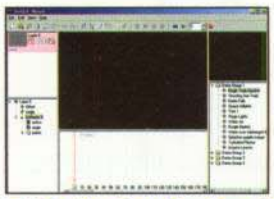
■ Otros juegos

Además, podéis encontrar un Tetris para todos los gustos llamado Rotoblox y un juego hecho con Blitz3d, nuestra herramienta: Platypus.

VÍDEO

Aquí os ofrecemos en esta ocasión dos excelentes programas de edición de vídeo:

■ Illusion 2.0



Programa de render, composición y animación para crear efectos espectaculares.

■ Media Studio Pro 6.5

Sencillo pero potente programa para editar vídeo de un modo sorprendente.

Zone of Fighters: de la cabeza al papel

El nacimiento de un videojuego empieza por su concepción. Todo comienza con la idea, la ilusión de hacer realidad el fruto que la imaginación va alimentando.

Hay que tener en cuenta que una cosa es ver el juego en la mente y otra muy distinta cómo será en la realidad. Se cae en el tópico de diseñar ideas que van más allá de las posibilidades técnicas que se poseen. Ocurre entonces que el resultado final no corresponde con el diseño original del juego y se tienen que realizar cambios sobre la marcha, retrasando el desarrollo.

Esta semana empezaremos el desarrollo de un videojuego estudiando la forma de llevar nuestra flamante idea al mundo real, de un modo práctico, factible y siempre dentro de nuestra capacidad de desarrollo.

Cuando tenemos la idea para un juego, lo primero que pensamos es de qué tipo será: acción, deportivo, aventura gráfica, puzzles, etc. Después, si será en 2D o 3D, y, por último, buscamos originalidad. No cabe duda de que la experiencia personal de cada uno como jugador aflora irremediablemente, y nuestras ideas pasan por los juegos que marcaron estilo. Es entonces cuando nuestro proyecto se parecerá a tal y a cual videojuego al que hemos jugado, o que hemos visto en Internet o en una revista especializada.

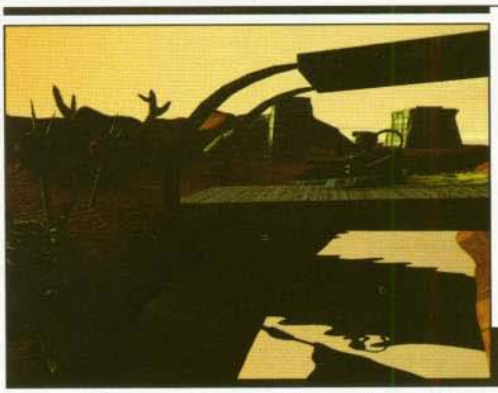
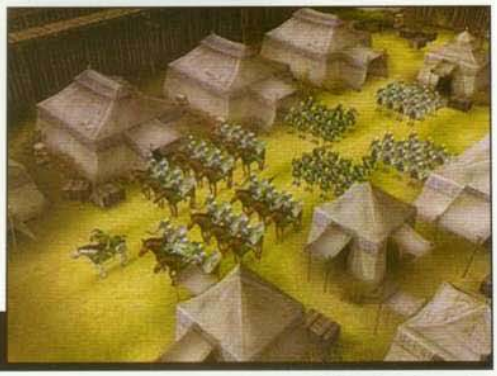


Imagen del aspecto del entorno 3D deseado para el juego Zone of Fighters.

Hoy en día, a pesar de la cantidad de nuevos desarrollos en el sector, no abundan mucho las ideas nuevas. Casi todos los juegos, aparentemente, son parecidos y tienen multitud de aspectos en común, sin embargo, siempre existe algún factor que los hace distintos uno de otros: nuevas armas, una apariencia gráfica diferente, diseño de la interfaz de uso, un sistema de audio más realista o un desarrollo de la jugabilidad poco común. Y es, en definitiva, la calidad y diseño de todos estos aspectos lo que hace que un videojuego sea mejor o peor que otro y, en contadas ocasiones, que pase a crear un nuevo estilo.



Los juegos de estrategia real como Civilization, Age of Empires o la serie Tycoon son siempre los más vendidos.

LA IDEA DEL JUEGO

Nuestra intención no es crear un estilo nuevo o marcar historia, simplemente pretendemos utilizar una idea ya existente, para facilitar un método pedagógico práctico y cómodo que ayude a la comprensión del complejo arte de desarrollar videojuegos.

Partiremos de la premisa de un juego de combate en el que el jugador se enfrenta a uno o a varios contrincantes. Estos adversarios pueden ser manejados por el ordenador o pueden ser otros jugadores conectados a través de Internet o la red local.

El juego transcurre en una gran superficie formada por montes, ríos, bosques y construcciones. En ella se encuentran gusanos subterráneos letales y plantas carnívoras, entre otros peligros. Los jugadores disponen de una nave de combate con posibilidad de utilizar distintos tipos de armas y escudo de energía para la protección. Se desplaza por el terreno flotando a cierta distancia del suelo, pudiendo acelerar y frenar a voluntad pero nunca parar del todo. Asimismo, pueden sumar puntos por medio de bonos espar-

NOTA

Los juegos de estrategia en tiempo real son el tema predilecto por las desarrolladores de videojuegos para ordenadores, dada su gran aceptación en el mercado.

3

Imagen de las urnas que contienen los terrenos de combate de Zone of Fighters.

cidos por la superficie o aniquilando enemigos, animales o plantas. Vencerá el último jugador que quede vivo.

Estos terrenos no forman parte de ningún mapa o región sino que se encuentran en el interior de enormes urnas sobre pedestales, la "zona de luchadores" (zone of fighters).

(■) LA HISTORIA

Como cada año, en el sistema Hooman, se celebra el mayor acontecimiento de la galaxia, la prueba de los campeones, el desafío definitivo, el combate mortal en grandes terrarios cargados de peligros llamados Zone of Fighters.

Los participantes, venidos de todos los rincones, lucharán entre sí en bio-naves de combate diseñadas para ser guiadas con la mente. El reto, sobrevivir. Vencerá el último que quede con vida. No hay piedad para los cobardes.

Bueno, ya tenemos a grandes rasgos la idea de nuestro juego. Y para completarla, nos falta definir algunos aspectos más.

(■) TIPO DE JUEGO

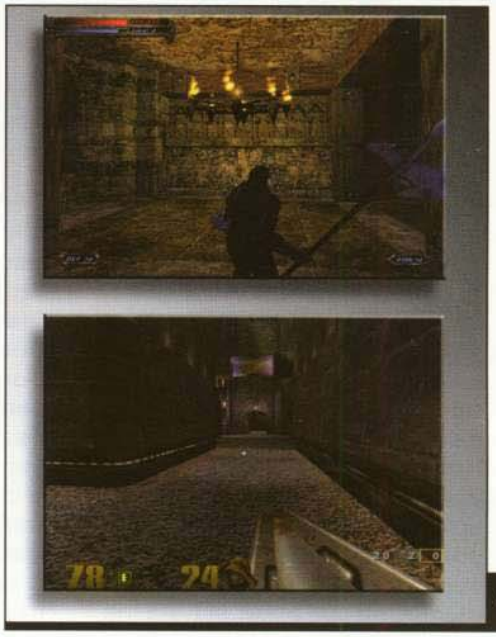
En este apartado decidiremos si el juego será en 2D o 3D y a qué estilo pertenecerá. Según el planteamiento de nuestra idea, la forma más real y espectacular de llevarla a cabo sería en un entorno 3D real. En definitiva, *Zone of Fighters* será claramente un juego de acción 3D.

Actualmente, multitud de juegos suelen mezclar los géneros. Así, es fácil encontrarlos de acción-aventuras, acción-puzzles o acción-velocidad; lo que diferencia a uno de otro es el desarrollo de la acción.

(■) SISTEMA DE VISUALIZACIÓN

Representar la acción de un juego por medio de un sistema de visualización en tres dimensiones ofrece multitud de ventajas, sobre todo al jugador, pero también ayuda a conseguir un mayor realismo. En este sistema, el jugador pasa a ver la acción como si observara a través de un cámara situada en distintos puntos. Desaparece pues la limitación que existe en un sistema de dos dimensiones, donde la acción se representa desde una sola vista y los gráficos siempre con el mismo ángulo: de un lado u otro, de frente o espalda o desde arriba o desde abajo. Con una dimensión más, es posible ver el gráfico y el entorno desde todos sus ángulos y en tiempo real.

En las ilustraciones se exponen dos ejemplos de posición de cámara. En la foto superior se muestra un juego con perspectiva en tercera persona y en la imagen inferior con perspectiva en primera persona (FPS).



4

Dos ejemplos de posición de cámara: en la foto superior se muestra un juego con perspectiva en tercera persona y en la imagen inferior con perspectiva en primera persona (FPS).

Cambiando la posición de la cámara o la vista del protagonista principal modificaremos también el tamaño y posición de todos los gráficos. Estos cambios hacen la distinción entre los juegos 3D. Por ejemplo, un FPS (*First Person Shooter*), o juego en primera persona, tendría situada la cámara en un plano subjetivo; es decir, el jugador nunca ve a su protagonista porque la cámara nos muestra lo que éste ve. Cuando se habla de

5

La perspectiva isométrica era la preferida de los antiguos juegos de estrategia en 2D.

NOTA

En 2D, en una perspectiva isométrica (perspectiva 1-1-1), todos los gráficos que componen el entorno *(tiles)* tienen forma de rombo, con todos sus ángulos de 45 grados iguales, creando así profundidad y obteniéndose una apariencia 3D.

un juego en tercera persona, la cámara está situada por encima y a la espalda del protagonista, como si una tercera persona estuviera viendo la acción. En ocasiones, los juegos en 3D se muestran con perspectivas utilizadas en juegos 2D, como la cenital (vista desde arriba) o la isométrica, pero con la particularidad de que, al ser 3D real, es posible girar la cámara alrededor del entorno.

En *Zone of Fighters* vamos a utilizar la posibilidad de cambiar la posición de la cámara a nuestro antojo; de esta forma obtendremos varias vistas del entorno. Daremos así la oportunidad al jugador de ver la acción desde la posición que desee, facilitando con ello la jugabilidad. Por medio de las teclas de función le ofreceremos al jugador una serie de vistas fijas y por medio del ratón la opción de mover la cámara libremente.

Otra cuestión importante a tener en cuenta es aumentar el rendimiento del juego. Se podrían utilizar un par de sistemas: controlando el nivel de detalle o LOD (*Low Of Detail*) o creando otro juego de texturas de menor tamaño para objetos lejanos donde el detalle de su apariencia no es tan necesario.

DEFINICIÓN

El rendimiento visual de un juego consiste en mantener una velocidad de *frames* (imágenes) por segundo aceptable para evitar saltos en la acción. Generalmente, una velocidad de 25 a 30 frames por segundo es lo normal para conseguir un resultado fluido.

DEFINICIÓN

El sistema LOD consiste en reducir el número de polígonos en objetos situados a mayor distancia de la cámara.

ESTILO GRÁFICO

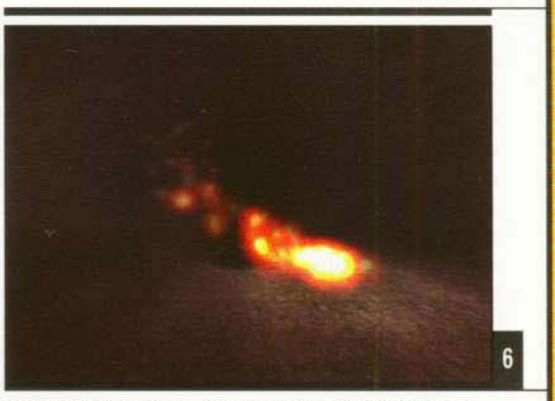
Es evidente plantear unos gráficos de acuerdo con la historia del juego. Si, por ejemplo, nuestro juego trata de batallas de guerreros medievales, los gráficos tendrán que ser propios de la época. Habrá que diseñar espadas y hachas en vez de pistolas láser.

Zone of Fighters se desarrolla en un mundo espacial imaginario sin ningún antecedente histórico, pero sí se sabe que existe una tecnología avanzada. Habrá grandes superficies al aire libre con vegetación, ríos, volcanes, nubes y sol. Habitarán seres de varias clases. Se dispone de naves de combate y distintos tipos de armas. Además, habrá esparcidas edificaciones como almacenes, generadores de energía y puentes. Toda esta apariencia gráfica puede ser de diseño libre. En este caso, le aplicaremos un poco de imagen futurista a los objetos artificiales y un toque de fantasía al entorno.

Ciertamente, a todo esto tenemos que añadirle los efectos especiales, los más reales y espectaculares posibles. Se incluirán efectos climatológicos como lluvia; también humo y fuego para los volcanes. Habrá disparos, explosiones y por último un poco de sangre para las criaturas letales que nos acompañarán en la acción. Para este cometido, utilizaremos un completo sistema de partículas que estudiaremos con profundidad en próximas entregas.

AMBIENTACIÓN SONORA

Un aspecto importante es sin duda el sonido del juego. Con él, vamos a reforzar la temática que hemos diseñado, creando el ambiente adecuado.



6

Ejemplo de un sistema de partículas simulando fuego y humo. En el curso veremos otros muchos efectos.

Para empezar, a cada acción que hagamos en el menú -con el ratón- le aplicaremos un sonido; así confirmamos al jugador su elección. Puede valer con el sonido de un *clic* o pequeño pitido. También sonará de fondo un tema musical con cierto ritmo pero de índole militar, invitando a la competición.

Una vez dentro del juego, los efectos de sonido se tornan más importantes, porque pueden ayudar en gran medida a resolver las situaciones. Es importante recordar que se debe aprovechar las características del sonido 3D para mejorar nuestros propósitos.

NOTA

Cada lector según vaya completando su formación a nivel gráfico y de programación podrá generar cambios en *Zone of Fighters* siguiendo su propio estilo y gusto.

DEFINICIÓN

Un sistema de partículas es una rutina especial que permite simular, por medio de pequeños gráficos, el movimiento de moléculas. Se utiliza para crear efectos de humo, fuego, dinámica de fluidos, etc.

Dotaremos a nuestra "bionave" de un sonido con tono ascendente en el momento de la aceleración y uno descendente en la frenada. Para el disparo haremos distinción dependiendo del tipo de arma que se utilice y para el escudo de energía, un sonido envolvente y algo denso, además de un pequeño aviso sonoro cuando su activación esté llegando a su fin. De este modo, el jugador dispondrá de tiempo para reaccionar, por ejemplo, en medio de un combate.

Aplicaremos otra clase de avisos sonoros en situaciones como: un nivel bajo de energía, la obtención de bonos o cualquier tipo de items (munición, arma, energía, escudo, etc.) Asimismo, en cualquier contacto detectado por parte de las bionaves con otro objeto del escenario (edificaciones, árboles, montañas u otros jugadores) se producirá la respuesta sonora correspondiente. No olvidemos los efectos sonoros, generalmente explosiones, que los impactos por disparos causan en la superficie, demás elementos del terreno u objetos.

Un detalle cada vez más extendido que ayuda mucho durante el desarrollo del juego es conducir al jugador por medio de una voz de fondo o *voz en off*, bien recordando el manejo de la bio-nave, avisando de la cercanía de peligros o en cada acción del jugador como elección de armas o acierto en el disparo.

En este tipo de juegos, el desarrollo puede volverse bastante activo, debido al combate y la lucha con los demás habitantes de la zona. Sería apropiado acompañar toda esta acción con música movida, con bastante ritmo, para suministrar aún más emoción al jugador.

DEFINICIÓN

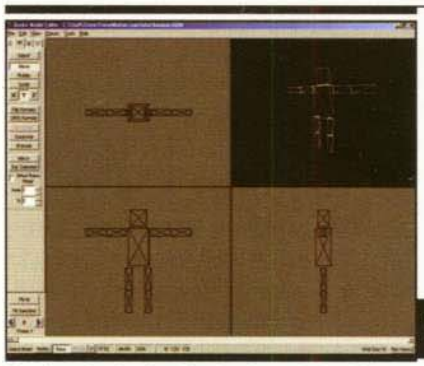
▸ ¿QUE ES UN MOD?

El concepto MOD fue inventado por ID Software para su juego *Quake* y consiste en tener acceso al código para poder así realizar cambios en él. De esta forma, los jugadores, convertidos en diseñadores y programadores, descubren nuevas formas de ampliar los juegos: desde añadir simples mapas, nuevas texturas o monstruos, hasta diseñar nuevas modalidades de juego como "capturar la bandera" o incluso permitir el nacimiento de un juego nuevo, como ocurrió con el *Half Life* o *Counter Strike*.

Hay dos modalidades de MOD: para un jugador o para multijugador. Los primeros, generalmente, sólo proporcionan extensiones de niveles o características nuevas, mientras que los segundos proporcionan nuevas modalidades de juego.

COMPLEMENTOS DEL JUEGO

Como recordaréis, en el tutorial del número anterior, comentábamos la necesidad de acompañar a nuestro juego de complementos que ayuden a obtener un mayor interés por parte del usuario. Precisamente, cada vez son más solicitados los editores de niveles. Generalmente, estas aplicaciones forman parte de las herramientas que los desarrolladores poseen para facilitar el trabajo. Suelen ser de diseño un tanto complicado y de uso muy personal, y en pocas ocasiones están preparadas para servir al público. El gran interés del jugador hacia este tipo de ampliaciones ha propiciado que las compañías hagan un desarrollo paralelo de sus herramientas para incluirlas en sus lanzamientos. Además de los editores de niveles, como *Q3Radiant* para la saga *Quake* o el *Worldcraft* para *HalfLife*, debemos sumar visualizadores de modelos, como el que ofrece el juego *Neverwinter Nights* o sistemas para crear tus propios *MODs*.



7

Un ejemplo de visualizador de modelos MD2.

En definitiva, en esta entrega hemos convertido un montón de ideas sueltas en un diseño tangible, real y práctico. Ahora nuestro proyecto de videojuego va adquiriendo forma, y la imagen que teníamos de él en la mente la hemos plasmado en papel, por escrito.

Ahora tenemos la tarea de diseñar cada elemento que compondrá nuestro juego, con más detalle. Realizaremos dibujos o bocetos de todos los aspectos del juego para facilitar el diseño gráfico. Describiremos el comportamiento de cada componente y el mecanismo de sus acciones. Fabricaremos nuestras propias herramientas de desarrollo como visualizadores de terrenos y objetos o editores de zonas de combate. Y por último llevaremos todo a la práctica, a la vida real, desarrollando el código.

En el próximo número...

... empezaremos a diseñar con más detalle los elementos que componen *Zone of Fighters*.

Herramientas para hacer los gráficos de un juego

La calidad gráfica de los videojuegos ha caminado siempre paralela al desarrollo tecnológico de la informática. No hace más de diez años, era imposible ver más allá de 16 colores en un PC y, prácticamente, el 3D se limitaba a líneas y modelos sin textura alguna.

Con la llegada del sistema operativo Windows de Microsoft, se abrió una nueva puerta para la evolución de la infografía que afectó inevitablemente al desarrollo de los videojuegos para PC.

Se consolidó aún más el PC como ordenador para uso multimedia y empezaron a aparecer tarjetas gráficas mejores al servicio de microprocesadores cada vez más potentes.

En un principio, las poderosas herramientas gráficas estaban a disposición de grandes producciones cinematográficas, la televisión o la publicidad; sin embargo, a medida que evolucionaban las capacidades multimedia de los ordenadores personales, los juegos también lo hacían y su desarrollo demandaba más y mejores paquetes infográficos.

Las primeras herramientas gráficas de uso doméstico se utilizaban para dibujar y manipular gráficos en dos dimensiones. Generalmente, constaban de una ventana o parte de la pantalla donde era posible realizar operaciones de dibujo, y de un grupo de iconos que representaban las herramientas disponibles para el usuario: pinceles, goma para borrar, selección de color y posibilidades de dibujar polígonos como rectángulos, círculos, etc.

Estos programas eran utilizados por los creadores de juegos para sus gráficos y existieron en sistemas domésticos anteriores a la maduración multimedia del PC. El más conocido fue el *Deluxe Paint* para los ordenadores Commodore y se convirtió en el precedente de los actuales programas de dibujo.

Con la llegada del 3D a los videojuegos, la demanda de herramientas gráficas aumentó considerablemente. Además del tradicional y necesario programa de dibujo y retoque fotográfico, era primordial utilizar aplicaciones para la creación de objetos poligonales. Aparece también la necesidad de dar apariencia a estos modelos 3D por medio de texturas. Estos elementos gráficos pueden ser dibujados fácilmente con cualquier programa gráfico 2D; aun así, emergen utilidades que permiten dibujar directamente sobre el objeto 3D, facilitando la labor del grafista.

El mercado empieza a inundarse de herramientas 3D al servicio del desarrollo de infografía lúdica, apareciendo una gran oferta en calidad y en precio para el profesional y el usuario de a pie.

¿QUÉ PROGRAMAS SE UTILIZAN?

Hoy día, las producciones de videojuegos han alcanzado una cota altísima de calidad gráfica y precisan de herramientas especializadas. A pesar del gran número existente de ellas, la mayoría cumple con los requisitos

1

Una imagen del legendario Deluxe Paint para ordenadores Commodore Amiga.

2

Photoshop es el programa de dibujo más utilizado en el mundo del diseño gráfico para videojuegos.

3

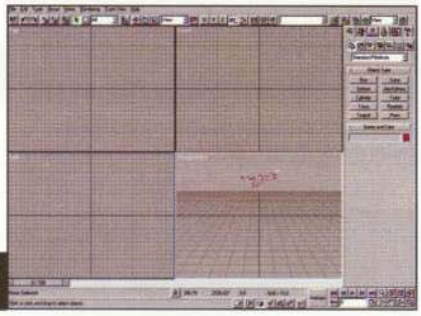
En la imagen, modelos del Commodore 64, uno de los ordenadores de 8 bits más famosos de la época.

NOTA

En los CD-ROMs que acompañan a esta obra se podrán encontrar demos totalemente operativas de todos los programas mencionados.

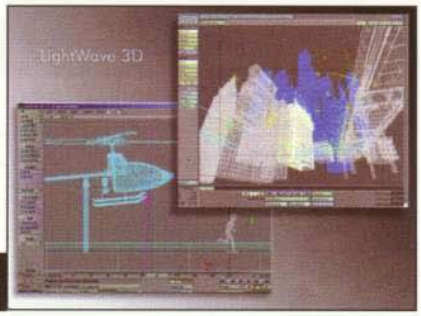
4

Paint Shop Pro 7 es ideal para realizar los gráficos de Zone of Fighters.



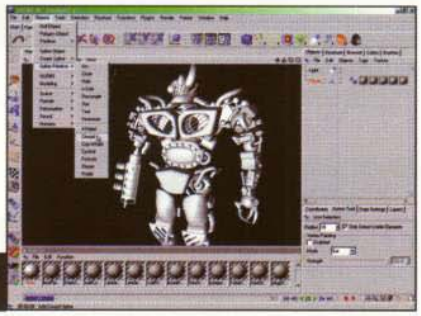
5

En la actualidad, 3D Studio Max de Discreet es el programa de diseño 3D más usado por los grafistas.



6

LightWave cada vez se utiliza más en la industria del videojuego. Es el favorito de ID Software.



7

Cinema 4D XL7 comparte cartel con LightWave en producciones televisivas y publicidad.

necesarios, y son su precio e implantación en el mercado lo que las diferencian. Debemos distinguir aquellas aplicaciones diseñadas para un uso en el campo de los gráficos en 2D y 3D.

GRÁFICOS 2D. DIBUJO Y RETOQUE FOTOGRÁFICO

Hay numerosas aplicaciones potentes para el diseño gráfico 2D, pero muy pocas han gozado de la fama necesaria para llamar la atención profesional y amateur. Sólo las grandes casas de desarrollo de software pueden competir en el mercado e implantar estándares. Tenemos, pues, el programa *Photoshop* de Adobe, *PhotoPaint* de Corel o el *Paint Shop Pro* de Jasc Software.

Adobe *Photoshop* es el líder mundial indiscutible y el preferido por los grafistas de videojuegos. Fue, prácticamente, el primer programa de edición y retoque fotográfico profesional del mundo y su diseño ha creado escuela gracias a su fácil interfaz de usuario, la gran potencia de sus herramientas, un sistema de composición por capas y su revolucionaria técnica para deshacer y rehacer acciones, entre muchas otras. Aun así, existe la competencia, y todas las demás aplicaciones poseen un diseño y potencia muy similares y son capaces de manipular todos los formatos de imágenes más comunes, luego pueden servir perfectamente para nuestros propósitos.

NOTA

A pesar de su bajo coste (unos 100 euros) Jasc *Paint Shop Pro 7* puede competir perfectamente en prestaciones con programas de lujo como Adobe *Photoshop*. Y se convierte en la herramienta 2D preferida del usuario medio.

GRÁFICOS 3D. MODELADO Y ANIMACIÓN DE OBJETOS

Rellenaríamos líneas y líneas enumerando los programas de diseño en tres dimensiones que existen, aunque actualmente en uso en el mundo de los videojuegos sólo hay unos cuantos. Al igual que en el diseño 2D con *Photoshop*, ocurre en el diseño 3D con el *3D Studio Max*. Este programa es ineludiblemente el líder mundial en el sector lúdico. Su arquitectura abierta ha posibilitado que sus características estén en continuo aumento y sus aplicaciones se extiendan para cubrir todas las necesidades que el diseñador gráfico de un juego precisa. Su formato de archivo genérico 3DS es un estándar mundial y rara es la aplicación o herramienta de programación 3D que no lo soporte. Es un programa ideal para modelar y animar objetos para juegos y también, por ejemplo, para generar películas animadas de presentación.

Hay otras aplicaciones 3D que se han adaptado perfectamente y con buena nota al diseño gráfico de juegos. Nos encontramos con herramientas de gran potencia utilizadas en el mundo del cine o la televisión, como *LightWave*, *Maya*, *SoftImage* o *Cinema 4D XL*, y otras también profesionales pero más asequibles como *Caligari TrueSpace* o *Bryce*. Esta misma editorial, Iberprensa, ha publicado recientemente un curso sobre Diseño 3D con *TrueSpace* y *Cinema 4D*.

En el ámbito profesional, las indiscutibles prestaciones de modelado y animación del *LightWave* y su relación calidad-precio, han contribuido a que pueda acompañar al *3D Studio Max* en el ranking de

herramientas más usadas en el desarrollo de videojuegos.

Cinema 4D XL7 también compite con las aplicaciones más utilizadas del momento, como *3D Studio Max* o *LightWave*, gracias a su robustez y versatilidad. Asimismo, la versión XL7 del *Cinema* incorpora el *BodyPaint 3D*, un sistema para el pintado de objetos muy potente.

Caligari, con sus versiones 4 y 5 del *TrueSpace*, ha encontrado un hueco importante en el diseño 3D para juegos, gracias a un acercamiento al usuario menos profesional -necesitado de una herramienta fácil de aprender, de bajo coste y con características competitivas-. Además, es de las pocas herramientas de diseño 3D que incorpora -de serie- la importación y exportación de modelos en formato DirectX (extensión de archivo .X), muy utilizado en el desarrollo de juegos para Windows.

Ha cautivado a diseñadores de todo el mundo gracias a su versatilidad, su potencia y su intuitiva interfaz de usuario. Integra todas las funciones necesarias para el modelado y animación de objetos 3D, ideales en el desarrollo gráfico para juegos.

El programa *Bryce*, creado por la desaparecida Metacreations, se ha convertido en un estándar en la creación de mundos artificiales y, hoy día, es extraño encontrar un grafista de videojuegos que no conozca sus fantásticas posibilidades para la producción de terrenos. Su singular interfaz de usuario lo convierte en una herramienta muy instintiva y fácil de aprender. Carece de la posibilidad de exportar objetos -a excepción de los terrenos 3D, que puede hacerlo en multitud de formatos-. También ofrece la posibilidad de guardar en disco una imagen del mapa de altura para incluirlo posteriormente en nuestros juegos.

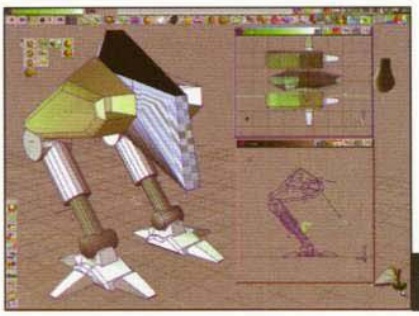
Muchos de estos programas pueden ser adquiridos sin demasiado esfuerzo económico por grandes desarrolladoras o incluso por las de nivel medio, pero resulta más complicado para el usuario doméstico o para un grupo pequeño que, a duras penas, encuentra mercado para sus juegos. Una solución a este problema la hallamos en aplicaciones 3D menos potentes, pero tremendamente útiles y económicamente aceptables como el *3D Canvas*, *MilkShape 3D* o el *Zmodeler*. Se pueden adquirir fácilmente a través de Internet y la mayoría de ellas están en los diferentes Cds que componen esta obra. Dado que hay infinidad de ellas, vamos a determinar cuáles se ajustan a nuestros propósitos.

NOTA

La facilidad de manejo del programa *Bryce* para crear terrenos es asombrosa. Su única y especial interfaz lo convierte en el creador de mundos más querido por los diseñadores gráficos. Es ideal para crear los terrenos para nuestros juegos, ya que permite exportarlos a multitud de formatos 3D, incluido DirectX, en su versión 5.

TEXTURIZADO DE OBJETOS 3D

Al margen de las aplicaciones 2D y 3D, se encuentran herramientas que mezclan los dos conceptos gráficos con el propósito de hacer más asequible, si cabe, el texturizado o apariencia de los modelos tridimensionales. Permiten importar objetos 3D para luego poderlos pintar con herramientas de dibujo de una forma cómoda e instinti-

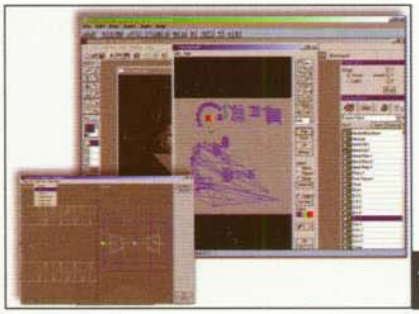


8

Caligari TrueSpace 5 es fácil de aprender y soporta todos los formatos 3D utilizados actualmente.

9

Bryce es un programa relativamente barato para la calidad que ofrece.

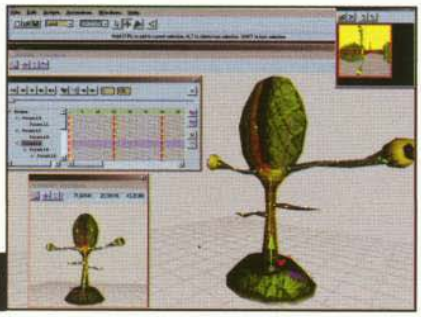


10

El Deep Paint 3D es una herramienta profesional para el texturizado de objetos 3D.

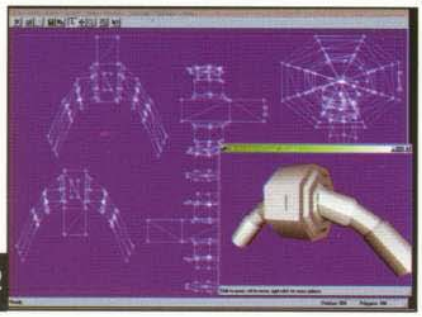
DEFINICIÓN

UV Mapping o Mapeado UV consiste en ajustar una imagen a una malla. UV es para la malla como las coordenadas XY lo son para la imagen. Las coordenadas XY de la imagen se ajustan a las UV de la malla, independientemente de la deformación que ésta sufra.



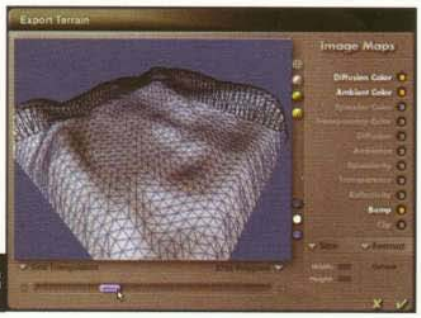
11

Con la aplicación Character FX podemos animar nuestro modelo con desenvoltura por medio de un esqueleto.



12

Además de crear plantillas para el texturizado de objetos y optimizar modelos 3D, LithUnwrap es freeware.



13

Antes de exportar el terreno, Bryce 5 proporciona la opción de optimizar el número de polígonos, mostrando en todo momento una vista del objeto.

va. Los más utilizados son el *Deep Paint 3D* y el *Painter 3D*.

¿QUÉ PROGRAMAS UTILIZAREMOS?

En primer lugar, vamos a enumerar las operaciones que llevaremos a cabo para el desarrollo de nuestros gráficos. De esta forma definiremos mejor las herramientas que necesitamos.

- **1.** Para todos los gráficos 2D, tales como iconos, pantalla de presentación, logotipo, etc., además de algunas texturas para los modelos 3D, nos hace falta un programa de dibujo y retoque fotográfico. Elegimos *Paint Shop Pro 7*. Es fácil de aprender y posee una potencia equiparable a programas tan famosos como *Photoshop*.
- **2.** Precisamos de un modelador que nos permita realizar objetos en 3D con pocos polígonos (baja poligonización) y que además pueda importar y exportar en la mayoría de los formatos de archivos que utilizan los videojuegos. El candidato más popular, más fácil de usar y el que más formatos maneja, es el *MilkShape 3D*.
- **3.** Después de fabricar al protagonista, necesitamos darle vida, animarlo. Aunque el *MilkShape 3D* permite la animación por medio de un esqueleto, resulta algo incómodo al manipular y depurar un gran número de movimientos. Por este motivo utilizaremos un estupendo programa llamado *Character FX*, que salva con creces este inconveniente de una manera realmente sencilla, utilizando un comodísimo editor de fotogramas llamado keyframer para manipular cada aspecto de la animación.

El secreto de un buen modelo 3D, aun teniendo pocos polígonos, es su apariencia final y esto se consigue por medio de una buena textura. Si bien este tipo de gráfico lo haremos con el *Paint Shop Pro*, para algunos objetos de nuestro juego vamos a utilizar el *Deep Paint 3D* como texturizador. Con él podemos pintar directamente sobre el modelo utilizando multitud de herramientas y, lo más importante, aumentaremos nuestros conocimientos.

Además, en su última versión, incorpora una estupenda utilidad para crear el UV mapping del modelo llamada *MercatorUV*. Hay otro aspecto del juego que falta, el terreno. Mediante programación se puede generar este tipo de superficies, pero necesitamos un diseño específico: un monte aquí, un río allá... y para ello tenemos que modelarlo o crearlo a medida. No hallamos un programa mejor para tal cometido que el Corel *Bryce 5* o cualquiera de sus versiones anteriores.

NOTA

***LithUnwrap* es un interesante programa que realiza un UV mapping de objetos 3D y genera las plantillas para poder dibujar, con un programa gráfico 2D, la textura de los modelos. Igualmente nos servirá para optimizar nuestros modelos reduciendo sus polígonos sin perder la forma.**

En el próximo número...

... empezaremos a conocer las aplicaciones gráficas que utilizaremos a lo largo de este curso. En el apartado 3D, nos introduciremos en el universo del *Milkshape 3D* y la baja poligonización, y en 2D, tomaremos contacto con *Paint Shop Pro 7*.

Formatos de audio para el juego

Los tipos de formato que utilicemos en el juego dependerán de la capacidad técnica que poseamos. Por tanto, es importante saber qué calidad de sonido queremos emplear. Cada uno de estos formatos posee unas características distintas de reproducción y un consumo de memoria diferente.

Vamos a explicar, en este capítulo, algunos de los formatos más comunes utilizados hoy día en el mundo de los videojuegos, cuál vamos a manejar en nuestro juego y la mejor forma de hacerlo.

FORMATO DE AUDIO WAV

Este formato fue creado por Microsoft para su sistema operativo Windows. Suele ser el más utilizado en el mundo multimedia por su flexibilidad. Al no llevar ningún tipo de compresión es ideal para transformarlo a distintos tamaños y frecuencias por medio de editores y codificadores de audio. Lo más destacado de este formato es la elevada calidad de audio que puede archivar, hasta una frecuencia de 44,1 KHz a 16 Bits en estéreo. Sin embargo, este rendimiento tiene un precio: el excesivo tamaño de los archivos que se generan y la cantidad de memoria que consumen.

El lector se preguntará, ¿en qué puede afectar al rendimiento de un juego el que la música, en formato WAV, tenga un fichero de gran tamaño? Pues bien, al cargar el archivo desde nuestro programa, lo hacemos directamente en la memoria. Un archivo WAV con una calidad máxima (44 KHz, 16 Bits Estéreo) prácticamente ocupa 10 Mbytes por minuto de audio. Así que, si nuestra música tiene una duración de 3 minutos, al cargarse en memoria ocuparía la terrible cantidad de 30 Mbytes, y eso sólo para la música. Para reducir el tamaño de los archivos WAV (independientemente de la duración) se suele mermar su calidad de forma aceptable, reduciendo su frecuencia.

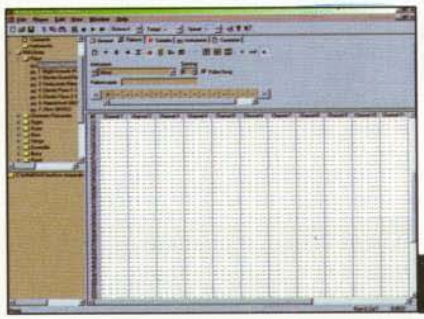
Generalmente, por este motivo, este formato se suele usar para los sonidos que irán en el juego y no para la música. Además, estas muestras se suelen reducir a calidades distintas dependiendo de su uso. En efectos especiales como disparos, explosiones, etc., es permisible una reducción de la frecuencia a 22 KHz, y para las voces incluso se puede prescindir del estéreo y convertirlo a mono, ahorrando así la mitad del tamaño que ocupa. Bajar hasta 11 KHz puede producir gran cantidad de ruido e incluso cortes en la reproducción, por lo que no es aconsejable.

No obstante, este formato de audio nos ofrece una gran ventaja: no necesita ser descomprimido, por consiguiente, no resta rendimiento al juego, un hecho éste que tendremos muy en cuenta a la hora de aplicar sonido a nuestro juego.

Existe una estupenda opción al problema del tamaño en *el formato WAV; y es utilizar* otro formato que comprima los datos sin perder prácticamente calidad. Nos referimos al MP3.

FORMATO DE AUDIO MP3

Este formato es el "rey" actualmente, debido a la reducción del tamaño de los archivos sin pérdida de calidad. Funciona por medio de algoritmos o fun-



1

En la imagen vemos el entorno de un secuenciador tracker.

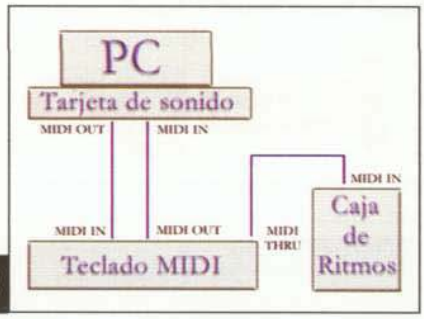
DEFINICIÓN

Frecuencia de muestreo es el número de mediciones de sonido que se realizan en un segundo. A mayor frecuencia, mejor calidad de audio obtendremos.

ORIGEN DEL MP3

Este sistema de compresión es la parte de audio del sistema de video MPEG y sus siglas vienen de MPEG-1 Layer 3. Anteriormente aparecieron los Layer 1 y 2 correspondientes a la cantidad de compresión posible y *que crearían los tres formatos* de audio MPEG:

- **MPEG Layer 1**
  Compresión de hasta 4 veces el original.
- **MPEG Layer 2**
  Compresión de 6 y 8 veces.
- **MPEG Layer 3 (MP3)**
  Compresión de 10 y 14 veces el original.



2

MIDI es un protocolo de comunicación digital estándar que permite conectar a dos instrumentos electrónicos de música entre sí o a un ordenador.

3

Mostramos los diferentes formatos que pueden intervenir en un juego y algunas de las calidades válidas por cada uno de ellos, de mayor a menor.

NOTA

8 ó 16 Bits, se refieren a la precisión con que se hace las medidas en el muestreo.
Si tenemos una resolución de 8 bits, la precisión que obtendremos irá desde 0 a 256 niveles posibles. Y si la resolución es de 16 bits, los valores posibles irán desde 0 hasta 65.535.

DEFINICIÓN

Un secuenciador tracker es un software utilizado para mezclar muestras de audio colocadas en distintas pistas (tracks) y obtener como resultado final un tema musical.

ciones que eliminan las frecuencias que el oído humano no puede percibir. Es capaz de mantener una calidad de disco compacto (44 KHz, 16 Bits Estéreo) reduciendo el archivo hasta 14 veces.

No podía faltar para toda esta maravilla un inconveniente, un sistema de descompresión para poder ser reproducido. Utilizar ficheros MP3 para la música o los efectos de sonido en nuestro juego provocaría una bajada del rendimiento, debido a que tienen que ser descomprimidos en tiempo real a medida que se reproducen. Asimismo, aumentaría los requisitos del ordenador.

Hay otras opciones para dotar a un juego de música con poco consumo de memoria y velocidad; son los ficheros de música MIDI, MOD, S3M o XM para secuenciadores.

FORMATO MIDI PARA MÚSICA

En este tipo de archivos podemos almacenar secuencias de música en forma de eventos MIDI. Ocupan muy poco espacio porque no poseen información de audio sino caracteres, más concretamente valores numéricos o parámetros propios del protocolo MIDI, como son el tipo de nota, la duración, etc. Estos valores son interpretados y provocan sonido a través de la tarjeta de audio del ordenador. Es fácil, pues, transportar la música de un sistema a otro. Además, estos eventos pueden modificarse desde el programa en tiempo real; y lo mejor de todo, no requieren un gran sistema para ser reproducidos. El inconveniente radica en que estos archivos no guardan el timbre, por lo que los sonidos empleados para la música están limitados a los que posee la tarjeta de sonido, y no todos los ordenadores poseen la misma.

También hay que decir que, aparte de no tener una calidad estándar, los ficheros MIDI no incluyen la reproducción de muestras, como *loops* de batería o voces, reduciendo la intención creativa. Para poder manejar secuencias formadas por muestras se puede utilizar los formatos MOD, XM o S3M.

FORMATOS MOD, S3 M Y XM DE MÚSICA PARA TRACKERS

Estos formatos son muy utilizados para proporcionar música a los videojuegos, debido a su relación tamaño-calidad-rendimiento necesario. Cada uno de ellos surgió de un secuenciador distinto y difieren unos de otros en prestaciones como número y calidad de las muestras, límite de filas por patrones, número de efectos permitidos, etc.

El formato MOD es el que posee más limitaciones, por ser el más antiguo; aun así, está muy extendido y es el más soportado por todos los secuenciadores. A continuación apareció el S3M, también muy popular y con algunas ventajas más. Con la llegada del programa *FastTracker* apareció el formato XM con muchas más posibilidades que sus antecesores. Pero de esta historia ya hablaremos con más detalle y realizaremos un mejor estudio de este estupendo secuenciador en posteriores entregas.

Hemos aprendido cuáles y cómo son los formatos de audio más utilizados. Ahora debemos escoger cuál de ellos vamos a incorporar en nuestro juego. De todas formas, aprenderemos a trabajar con todos ellos para tener así un mejor criterio a la hora de elegir. Antes de ello, desvelaremos las herramientas más utilizadas para fabricar el sonido de un videojuego.

En el próximo número...

... conoceremos las herramientas más utilizadas para componer el sonido de un videojuego y cuál utilizaremos nosotros para los efectos especiales y música de *Zone of Fighters*.

El editor y entorno de Blitz 3D

Todo lenguaje de programación moderno que trabaje bajo el entorno Windows –como es el caso del Blitz 3D– precisa de algo más que un procesador de textos para escribir código. Necesita un completo entorno de programación que además compile, ejecute y depure los programas.

Ciertamente, el corazón del Blitz 3D es una herramienta de estas características que ofrece una interfaz amigable, fácil de usar y muy potente. Además, incluye un sistema de ayuda interactivo completísimo al estilo HTML con multitud de descripciones y ejemplos.

En este número aprenderemos a usar todas sus características de una forma práctica.

¿CÓMO EMPEZAR CON EL EDITOR?

Una vez ejecutado el Blitz 3D (B3D) aparecerá el menú principal del archivo de ayuda (help). En realidad, este menú está en formato HTML y se navega por él como si de una página web se tratara. A través de los hipervínculos (links), en color amarillo, se accede a todas las secciones.

1

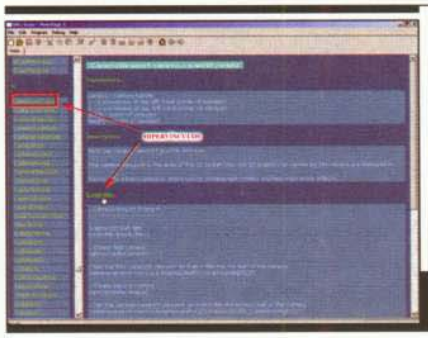
Un menú de ayuda en formato HTML nos da la bienvenida al entorno del Blitz 3D.

Además, para facilitar la navegación, el editor incorpora tres sencillos botones en la barra de iconos situada debajo de los menús desplegables. El botón sirve para ir al *index* y las flechas navegar hacia delante o atrás, para ir a las páginas de ayuda ya vistas.

Si pulsamos en "samples" podemos cargar desde disco, directamente al editor de programas, cualquiera de los ejemplos que nos proporcionan. En "Command Reference" encontramos la descripción y uso de todos los comandos del B3D y en "Languaje Reference" una descripción del funcionamiento del lenguaje.

Antes de continuar, vamos a disfrutar por un momento probando uno de los ejemplos del B3D. En primer lugar pulsamos en "samples", aparecerá una ventana del Windows con una lista de todos las carpetas de ejemplos. Elegimos el ejemplo "dragon" que está dentro de la carpeta */dragon*, pulsamos el botón de *abrir* e inmediatamente aparecerá el editor de programas con el código del ejemplo. Como se aprecia, en la barra de herramientas han aparecido algunos iconos más. No hay que preocuparse si no se entiende nada del código, pronto se irán explicando qué significan todas esas palabras.

De cualquier manera se puede ejecutar el ejemplo de tres formas distintas. La primera es pulsando el botón , si dejamos el puntero del ratón sobre el icono durante un segundo aparecerá escrito la acción que efectúa. La

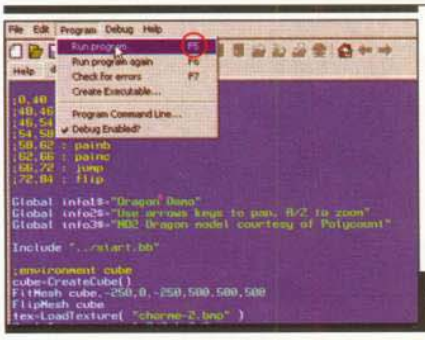


2

Todo los elementos de la ayuda se comunican entre sí por hipervínculos.

segunda forma es yendo al menú principal, desplegando "Program" y eligiendo la opción "Run program". Y la tercera es pulsando "F5"; evidentemente, es la más fácil.

El entorno del B3D, puede tener en el editor más de un programa a la vez. Vamos a cargar otro ejemplo. Para ir de nuevo al índice de la ayuda, podemos pinchar en el icono , pulsando en la pestaña "Help" del editor o mediante CTRL+H. Después de pulsar en "samples" de nuevo, elegimos el ejemplo "tron" de la carpeta *\tron*. Observamos que el anterior programa, *dragon.bb*, sigue en el editor además del nuevo. Para volver a él, pulsamos en la pestaña con el nombre "dragon.bb" o mediante CTRL+SHIFT+TAB



3

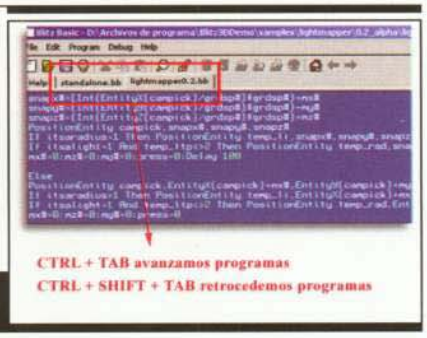
La manera más rápida de ejecutar un programa es pulsando F5.

una vez, o CTRL+TAB dos veces.

Es posible cargar cuantos programas se quiera y disponer de todos ellos en el editor. Esto nos ayudará cuando decidamos dividir el programa principal en otros subprogramas más pequeños para facilitar la corrección de errores o una posible ampliación. Un ejemplo práctico de este procedimiento, sería tener un fichero con el programa principal de nuestro juego de marcianos y otro fichero, con un programa que manipulara sólo el control de la nave por el jugador. Y para que todo funcione, el primer programa (el principal), incluiría una llamada al segundo.

NAVEGAR POR EL LISTADO

En primer lugar, vamos a cerrar los dos programas que hemos cargado en el editor. Nos situamos en cada uno de los listados y pulsamos sobre el icono o pulsando CTRL+F4. Aunque lo más rápido es ir al menú "File" y elegir la opción "Close All". Para aprender a editar nuestro listado, vamos a cargar el ejemplo "insectoids". El primer ejercicio que vamos a realizar es navegar por el listado. Mediante el ratón podemos colocar el cursor en cualquier lugar del texto, situando la flecha del ratón y haciendo clic con el botón izquierdo. Si lo que queremos es movernos poco a poco utilizaremos los cursores.



4

Es posible cargar al mismo tiempo todos los programas que queramos y acceder a cada uno de ellos rápidamente.

Para situarnos rápidamente al principio o al final del listado utilizaremos CTRL+INICIO (Begin) o CTRL+FIN (End). Una página de listado corresponde a una pantalla, así que para avanzar o retroceder pantallas lo haremos con las teclas RE PAG (retroceder) y AV PAG (avanzar). También podemos desplazarnos utilizando la barra de desplazamiento situada a la derecha de la ventana del editor, que aparecerá cuando el listado supere más de una pantalla de longitud.

Vayamos ahora a conocer una manera más rápida de movernos por los elementos del listado. En primer lugar situémonos en la primera línea. Si pulsamos la tecla FIN (End), el cursor se situará al final de la línea y con INICIO (Begin) al principio. Para recorrer cada palabra o símbolo utilizaremos CTRL+cursor derecho para avanzar y CTRL+cursor izquierdo para retroceder.

Hay distintos elementos que pueden formar parte del código de un programa como, por ejemplo, distintas funciones, tipos o subrutinas. Para desplazarnos rápidamente a cualquiera de estos componentes, disponemos de una ventana en la parte derecha del editor con un botón por cada uno de ellos: "funcs" para las funciones, "types" para los tipos y "labels" para las subrutinas.

Pulsemos el botón "funcs". Nos aparecerá una lista con todas las funciones que hay en el programa. Al elegir la función "CreateExplosion", el cursor se sitúa al principio de dicha función. Lo mismo ocurre con las otras dos opciones.

EDITAR EL LISTADO

Seguiremos, aprendiendo a seleccionar, mover, cortar, copiar y pegar partes del listado, así como buscar o reemplazar elementos.

5

Un sistema rápido para acceder a funciones, estructuras y subrutinas.

SELECCIONAR Y MOVER

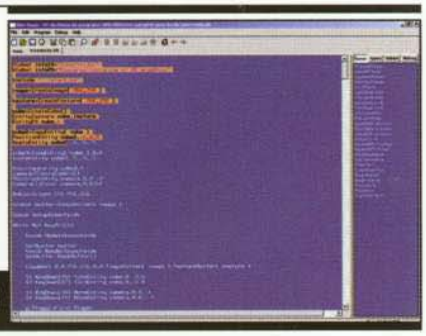
Se puede seleccionar usando el ratón o por medio del teclado. Generalmente, cuando se escribe código, es más rápido y útil cambiar el ratón por el teclado para no perder la atención. De todas maneras es bueno conocer perfectamente los dos procedimientos.

Vamos a seleccionar con el ratón desde el principio hasta donde se ve en la figura 6.

Debemos desplazarnos hacia abajo con el botón izquierdo pulsado, hasta el comienzo de la última línea a elegir y a continuación nos desplazamos hacia la derecha hasta el último carácter que queremos (recuerda que nunca dejes de pulsar el botón izquierdo en tus desplazamientos). Si nos equivocamos, siempre podemos deseleccionar todo, haciendo clic con el botón izquierdo.

También es posible seleccionar un sólo elemento, colocándote encima y haciendo clic con el botón izquierdo, o bien la línea completa haciendo doble clic.

Muy bien, ahora utilizaremos el teclado para esta operación. Deseleccionamos todo y nos colocamos al principio del listado con CTRL+INICIO (Begin). Realizaremos la misma selección que hicimos con el ratón. Nos desplazamos con los cursores manteniendo pulsada la tecla SHIFT, primero hacia abajo y luego hacia la derecha, hasta llegar al punto final de la selección. Si queremos deseleccionar basta con pulsar únicamente cualquier tecla del cursor.



6

Podemos seleccionar fácilmente uno o varias líneas.

Podemos comprobar la simbiosis perfecta que hay entre la tecla SHIFT y las demás teclas para movernos por el listado. Probemos, por ejemplo, a seleccionar una línea completa.

Nos situamos al comienzo de cualquier línea, pulsamos SHIFT+CTRL+FIN y podremos ver como el cursor viaja hacia el final (CTRL+FIN), pero al estar pulsando también SHIFT, realizamos la selección. Hagamos otro ejercicio. ¿Cómo podemos seleccionar todo el listado rápidamente combinando teclas?... ¡Exacto! Situándonos al principio del listado y pulsando SHIFT+CTRL+FIN. (Hay que recordar que si se presiona cualquier tecla del cursor se deseleccionará todo). Por supuesto, hay otra manera mejor de seleccionar todo el listado y es mediante la opción "Select All" en el menú "Edit" o con CTRL+A.

Hay un procedimiento muy interesante que permite desplazar partes de un listado de un lugar a otro por medio del ratón. Una vez hecha la selección (bien con el teclado o con el ratón), mantenemos pulsado el botón derecho del ratón y nos desplazamos. Observamos, que junto a la flecha del ratón aparece un cuadradito y el cursor se vuelve de color negro. Esto nos indica que tenemos vía libre para colocar la selección a continuación de dicho cursor. Este sistema es muy útil, pero algo complicado de manejar. Si cometemos algún error podemos deshacer la acción pulsando CTRL+Z.

TRUCO

Es posible moverse por el listado en cualquier dirección, desplazando el ratón con el botón central pulsado, siempre que lo tengas configurado correctamente en tu sistema operativo.

CORTAR, COPIAR Y PEGAR

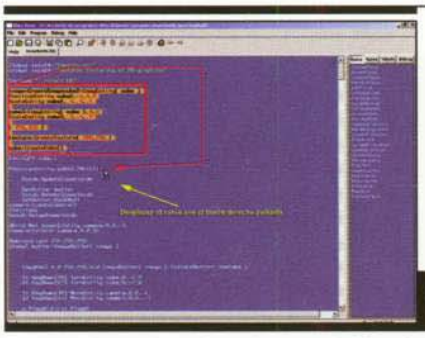
Veamos cómo *cortar* o *borrar* partes de un listado. Una vez seleccionado lo que queremos borrar, podemos hacerlo de varias formas: pulsando sobre el icono, con la tecla SUPR (suprimir) (DEL) o la tecla BACKSPACE (borrar), seleccionando la opción "Cut" del menú "Edit" y por último, con CTRL+X.

Si nuestra intención es *copiar* trozos de listado de un lugar a otro, debemos seleccionarlo, hacer una *copia* en la memoria pulsando el icono, eligiendo la opción "Copy" en el menú "Edit" o pulsando CTRL+C y posteriormente colocar la copia (*pegar*) en el lugar donde situemos el cursor, pulsando el icono, eligiendo la opción "Paste" del menú "Edit" o pulsando CTRL+V.

BUSCAR Y REEMPLAZAR

La opción "Buscar" del menú "Edit" (icono o CTRL+F) nos ayudará a encontrar elementos en nuestro listado; bien una palabra, un signo o una expresión.

Vamos a *buscar* sólo la palabra "texture" en el listado. Pulsamos CTRL+F, aparecerá una ventana con dos opciones. En la entrada para texto, escribimos "texture" y activamos la opción "sólo palabras completas" y aceptamos. En el listado se señala la primera palabra completa "texture" independientemente si contiene o no mayúsculas. Para encontrar la siguiente, basta con pulsa F3. Si desactivamos la opción "sólo palabras completas", buscará la coincidencia "texture" en cualquier lugar del listado.



7

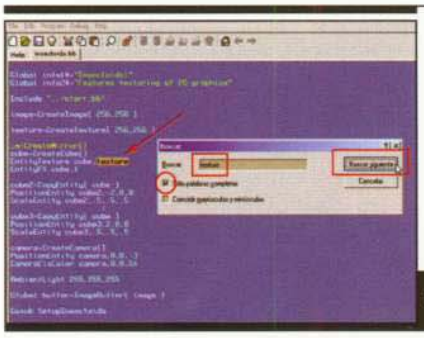
Ejemplo de desplazamiento de partes del listado con el ratón.

En el siguiente ejercicio, cambiaremos sólo la palabra "cube" por "CUBO" con la opción "Reemplazar" del menú "Edit" (CTRL+R). Aparecerá una ventana con dos entradas para texto.

En la primera escribiremos la palabra que queremos sustituir ("cube") y en la segunda la palabra sustituta ("CUBO"). Podemos sustituir sólo una palabra "cube" en concreto, buscándola con el botón "Buscar siguiente" y luego con "Reemplazar". Si por el contrario deseamos sustituir todas las coincidencias pulsamos en "Reemplazar todo".

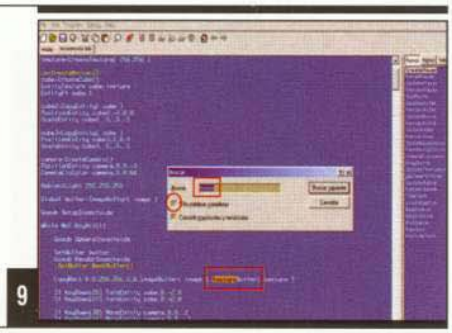
EJECUTAR, COMPILAR

Como vimos al principio, para ejecutar un programa basta con pulsar F5 y para ejecutarlo de nuevo F6. Si lo único



8

Con la opción de búsqueda se puede encontrar cualquier expresión en el listado.



9

Utilizando las búsquedas ganaremos mucho tiempo.

que queremos es compilarlo para buscar errores, pulsamos F7. Una vez que acabemos nuestro programa y éste funcione correctamente, podemos crear un fichero ejecutable independiente del editor con la opción "Create Executable" en el menú "Program".

EL DEPURADOR

El editor del B3D incorpora un depurador de código o "debugger" que nos ayudará a chequear la ejecución de nuestro programa paso a paso, permitiéndonos ver en cada momento el estado de cada variable y controlar el flujo del programa en tiempo real de una forma realmente sencilla. Este sistema de depuración funciona eficazmente gracias a la posibilidad de ejecutar programas en modo ventana que ofrece B3D.

Trabajar con el depurador activado implica un menor rendimiento en la ejecución de nuestro programa y ciertas reglas para activar los modos gráficos. Para activar o desactivar su funcionamiento basta con pulsar en la opción *Debug Enabled?* en el menú *Program*.

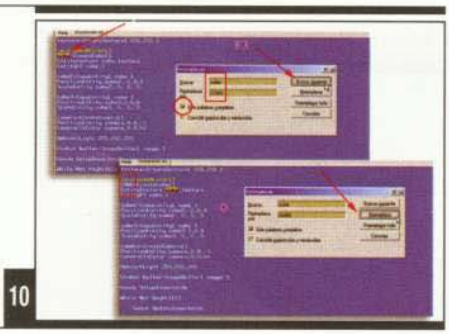
En la ventana de la derecha del editor tenemos el botón "debug" junto a "labels". Este botón nos mostrará una relación de todas las constantes y variables globales y locales que intervienen en el programa en curso. Al pulsar sobre el signo "+" que aparece junto a las palabras "Consts" y "Globals" se abre la lista completa de estas variables y sus valores. Al ejecutar el programa en el modo ventana podemos controlar estos valores en el curso de la ejecución y, así, comprobar los cambios que sufren. Sin embargo, estos cambios no se actualizan automáticamente en tiempo real en la lista mostrada en la ventana. Es preciso parar la ejecución del programa pulsando en el icono (/* insertar icono semaforo_rojo.jpg). Una vez pulsado vemos los cambios pertinentes en las variables afectadas. Además, en el listado, el cursor apunta a la última instrucción ejecutada en el momento del paro del programa y señala toda la línea en curso. Para continuar la ejecución pulsamos en el icono (/*isertar icono semáforo_verde.jpg).

El depurador tiene algunas opciones más que nos permiten recorrer la ejecución del programa paso a paso y de varias formas diferentes. Pero esto lo explicaremos en otra ocasión, cuando tengamos que ejecutar nuestro propio código.

Todavía nos queda aprender el sistema de depuración de código que tiene el editor del B3D, pero esto será cuando empecemos a escribir nuestro propio programa. El "debugger" o depurador de código, nos ayudará a chequear la ejecución de nuestro programa paso a paso, permitiendo ver en cada momento el estado de cada variable y controlar el flujo del programa en tiempo real de una forma realmente sencilla. Este sistema de depuración funciona eficazmente gracias a la posibilidad que admite Blitz 3D para ejecutar programas en el modo de ventana.

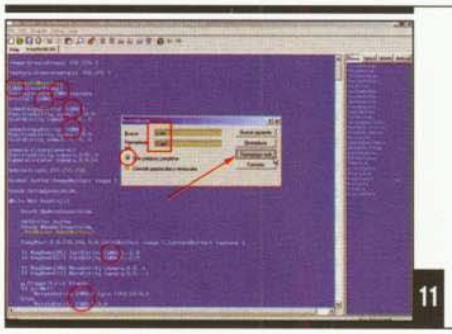
Aprenderemos más detalles de su funcionamiento cuando empecemos a escribir nuestro propio programa.

Antes, nuestra misión más importante es prepararnos para programar en Blitz 3D y para ello debemos conocer sus reglas de sintaxis, comandos y arquitectura de diseño.



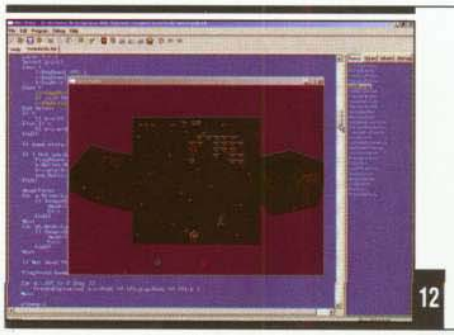
10

Procedimientos para la utilización de las opciones de reemplazar.



11

Otro ejemplo del uso de las opciones de reemplazar.



12

Un ejemplo de ejecución en modo de ventana.

En el próximo número...

... empezaremos nuestra misión de prepararnos para programar en Blitz 3D y para ello debemos conocer las reglas de sintaxis, comandos y arquitectura de diseño.

Editor de niveles Q3Radiant (I)

Incluido en el CD-ROM 1

ID Software con su juego *Quake* introdujo en el mundo del desarrollo de videojuegos un nuevo sistema de visualización de mapas 3D, basado en la técnica de árboles BSP, para representar sus niveles.

Este sistema fue adoptado rápidamente por la mayoría de desarrolladoras y se ha convertido prácticamente en un estándar muy usado para juegos 3D en primera y tercera persona. Multitud de motores 3D comerciales soportan este tipo de formato y resulta muy recomendado para diseñar nuestros mapas de interiores.

Q3RADIANT

Este editor de niveles fue desarrollado por ID Software como herramienta para sus diseñadores de mapas. Posteriormente lo hicieron público, y en la actualidad es uno de los más utilizados por la comunidad. No es precisamente el más fácil de usar, pero es versátil, estable y potente.

DEFINICIÓN

Un árbol BSP *(Binary Space Partitioning Tree o árbol binario para la partición de espacio)* es básicamente una forma de organizar objetos dentro de un espacio. Esta técnica aplicada a un entorno 3D coloca los objetos en una estructura de árbol la cual es seguida por el programa de forma ordenada y decide qué polígono se debe mostrar y cuál no. Es un buen método para eliminar del campo de visión áreas ocultas.

En este tutorial, dividido en dos partes, aprenderemos cómo usar los recursos básicos de este editor. Y La mejor manera para empezar es utilizando la práctica, así que comenzaremos fabricando una sala con algunas estructuras añadidas como columnas, puertas, etc.

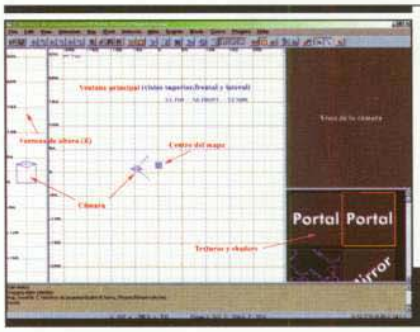
LOS PRIMEROS PASOS

En primer lugar, debemos tener instalado el *Q3Radiant* y el *Quake III* en nuestro disco duro, además de una tarjeta aceleradora.

Al ejecutar el programa, lo primero que vemos son distintas áreas divididas entre sí con una parrilla o *grid* de fondo (ver figura 1). La configuración de los colores de estas áreas la encontramos en "Colors", la ventana desplegable de "Misc" en el menú.

Para trabajar con *Q3Radiant* es aconsejable utilizar el acceso corto a las operaciones por medio del teclado. En "Command List" situado en "Help" del menú del editor encontraremos la lista de comandos. Es enorme, pero tampoco es necesario que la asimiles toda de una vez.

Inicialmente, te encontrarás en la vista XY superior (Top). Con *CTRL + TAB* se puede cambiar. En el centro hay dos objetos de color azul. El cuadrado con una "Z" es el "Chequeador-Z", que indica el centro del mapa, y el segundo es la cámara. Para desplazar la cámara en el área de vistas se utilizan las teclas del cursor y para moverla hacia arriba o abajo en el área de alturas, *CTRL + ratón + botón izquierdo*. Para mover la parrilla trasladamos el ratón pulsando el botón derecho.



1

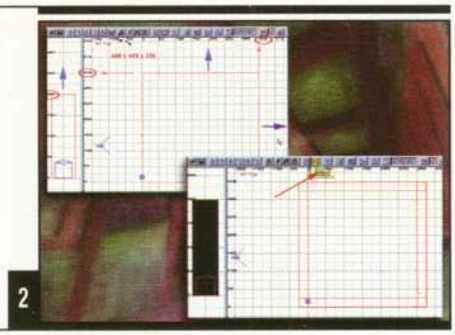
LA ESTRUCTURA PRINCIPAL DE LA SALA

Comencemos nuestra sala ajustando el tamaño de la parrilla a 64. Lo haremos pulsando el *5* del teclado no numérico (las opciones están entre 1 y 7). En "Preferences" situado en el desplegable "Edit" del menú activamos la casilla "Snap T to Grid" para ajustar todas las piezas que dibujemos a las divisiones de la parrilla.

Para crear el suelo de la sala, dibujamos un cuadrado en la ventana principal, de 448 x 448 unidades desplazando el ratón hacia la derecha manteniendo pulsado el botón izquierdo (el

NOTA

Los mapas del *Q3Radiant* se dividen en dos clases de componentes: los *brushes* o pinceles y las *entidades*. Los primeros representan toda la geometría del mapa, es decir, los bloques que forman las construcciones, que es lo que nos interesa, y los segundos constituyen el resto de elementos que forman el juego Quake III, como armas, munición, efectos especiales, luces, etc. (Se representan en el editor como bloques coloreados).


2

cuadrado estará formado por líneas discontinuas de color rojo). Si has tenido problemas en obtener las dimensiones exactas, puedes agrandar o estrechar el suelo colocando el puntero del ratón en el exterior de la línea de uno de los lados del cuadrado y desplazando el ratón pulsando el *botón izquierdo*. Si deseas mover el suelo entero, desplazas el ratón + botón izquierdo con el puntero en el interior del cuadrado.

Una vez que tenemos el suelo con las dimensiones exactas, vamos a fabricar las paredes y el techo de la sala. En la ventana Z (de alturas) alargamos el cuadrado hasta 256 unidades; así nuestra pieza quedará con las medidas 448 x 448 x 256. A continuación haremos un "hueco" (hollow) dentro del cubo que hemos fabricado pulsando sobre el botón "Hollow".

Para terminar la estructura pulsamos *ESC*. Nuestro cuadrado se volverá de color negro. Si movemos la cámara hacia el centro del cubo observaremos en la vista de cámara que estamos dentro de una habitación.

APLICANDO TEXTURAS

Para añadir texturas propias al *Q3Radiant* debemos copiar nuestra carpeta de archivos de texturas (en este caso la carpeta "texturas_tutorial" del CD) en el directorio "textures" que se encuentra en la carpeta "baseq3" dentro de "quake3". Una vez realizada la operación, entramos de nuevo en el programa y seleccionamos "Textures" del menú, luego "Load" y de nuevo el "Load"que aparece dentro de éste. A continuación, elegimos el directorio "Textures" y aceptamos.

Seguidamente tendremos que elegir de la lista de texturas la nuestra. De nuevo en "Textures" de la barra de menús elegimos "Load" y posteriormente "Load from list"; de la lista elegimos "texturas_tutorial" y "Load". Listo, las imágenes de la lista aparecerán al pulsar sobre la ventana de texturas.

Ya estamos preparados para asignar a cada parte de la sala nuestras texturas. Vamos a seleccionar una de las paredes. Para hacerlo, basta con elegirla en la vista de cámara o en la ventana de edición con el ratón, pero manteniendo pulsada la tecla *Shift*. La pared se volverá de color rojo en todas las ventanas. Para deseleccionarla se puede hacer pulsando *ESC* o de nuevo con *SHIFT + Clic ratón*. Seleccionamos todas las paredes y pulsamos sobre la textura de ladrillos. El aspecto de las paredes cambia y se vuelve enladrillada. Para el suelo elegiremos la textura de tablas de madera, aplicamos y pulsamos *ESC*, y para el techo la de la franja gris.

Q3Radiant está provisto de una herramienta que modifica todas las cualidades de los brushes o elementos geométricos del mapa llamada "Surface Inspector" (inspector de superficies), la cual aparece pulsando la tecla "S". Con esta herramienta vamos a aumentar el *tileado del techo verticalmente* y girar la textura 90 grados. Con el "Surface Inspector" activado, seleccionamos el techo. Observamos que "Vertical Strech" está a 1.0000; pues

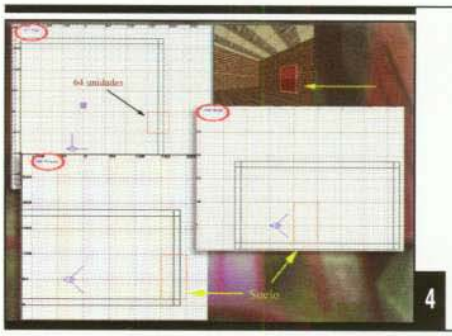
3

bien, lo aumentamos poco a poco hasta llegar a 1.8000 (podemos ver cómo la textura del techo se va estrechando). A continuación "Rotate" lo aumentamos hasta 90 (la textura gira 90 grados)

Antes de terminar esta primera parte, vamos a hacer una puerta con la función "CSG Subtract".

Pulsamos ESC. Dibujamos en la vista XY Top un cuadrado en una de las paredes. Cambiamos de vista (*CTRL+ TAB*) y ajustamos en tamaño y posición la puerta. A continuación pulsamos el botón "CSG Subtract".

Borramos el cuadrado con la tecla "Borrar" *(BACKSPACE)* y forma un hueco en la pared. En realidad "CSG Subtract" realiza una operación booleana de restar entre la pared y la caja que le hemos incrustado; es decir, elimina todo lo que está en contacto entre las dos piezas.


4

NOTA

Las texturas deben estar en formato targa (TGA) de 24 bits de color (sin canal alfa) o 32 bits (con canal alfa) También son válidas en formato JPG a 24 bits de profundidad de color.

En el próximo número...

... añadiremos los elementos de la sala como columnas, una lámpara con su luz y una chimenea encendida.

Los videojuegos entran en casa

El verdadero *boom* de los videojuegos surgió cuando estos comienzan a entrar en los hogares de todo el mundo. Paralelamente al éxito de los primeros arcades como *Pong* o *Space Invaders*, nacía el afán de la industria electrónica por llevar esos éxitos al ámbito doméstico. Apareció entonces la consola *Odyssey* de Magnavox, con una versión del *Pong* llamado *Ping-Pong*, tan simple que ni siquiera tenía marcador. Sin duda, fue la compañía Atari quien realmente triunfó con una versión del *Pong* más perfecta...

LOS PRIMEROS JUEGOS EN CARTUCHO

Realmente, estas cajas electrónicas no se podrían denominar videoconsolas, sino videojuegos en sí mismos, ya que sólo disponían de un juego grabado en un chip. Hasta la llegada de los juegos intercambiables contenidos en cartuchos no surgió el significado de videoconsolas. Este sistema apareció por primera vez con la consola *Fairchild Channer F*, pero fue de nuevo Atari quien revolucionó el formato cartucho con su modelo *2600 Video Computer System*. Sus juegos no tenían tanta calidad como los de *Intellivision* o *Colecovisión;* sin embargo, permitían la posibilidad al usuario de elegir entre distintas variaciones del juego, alargando su vida y su jugabilidad.

Por aquel entonces, aparecían en los hogares los primeros microordenadores personales de 8 bits -como el ZX-Spectrum o Commodore VIC20- más potentes para jugar. No obstante, la gente seguía prefiriendo las consolas por su sencillo uso y la cantidad de cartuchos que había. Además, triunfaban los arcades de recreativas como *Space Invaders, PacMan, Galaxian, Missile Command* o *Tank* y las consolas brindaban la oportunidad de jugar a buenas versiones de estos juegos cómodamente ante el televisor. Fue *Space Invaders* el primer juego de recreativas que se licenció para ser introducido en un cartucho de la Atari 2600, la cual no duró mucho en obtener toda la exclusividad de los demás éxitos.

NOTA

La magnífica videoconsola 2600 de Atari revolucionó el concepto de jugabilidad tanto en diseño exterior como en los juegos. Permitía seleccionar entre televisión en color y blanco y negro, así como controles individuales para cada jugador para cambiar las características del juego. Podía conectar distintos sistemas de mandos como joystick o ruedas.

NINTENDO Y SEGA

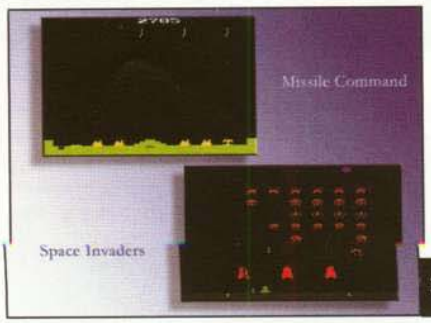
A finales de los 70 y principio de los 80, Atari era el líder indiscutible y muchas empresas fabricaban cartuchos de juegos para sus consolas. A partir de ahí empezó el declive de la empresa debido a la mala gestión de los negocios y el gran número de juegos malos que no se vendían. Aunque Atari fabricó algunos sistemas más con juegos mejores, fue la compañía nipona Nintendo la que cogió el testigo del éxito. Lanzó al mercado la

1

Imágenes de la consola 2600 de Atari, toda una revolución en su momento.

2

Atari 2600 introdujo los primeros cartuchos con posibilidad de variaciones distintas del juego. Un gran logro técnico y comercial.

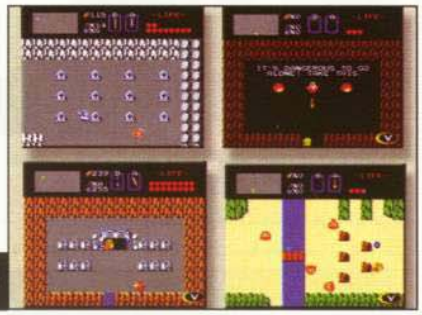


3

Missile Command y Space Invaders fueron versionados para la consola con gran éxito, a pesar de ser técnicamente inferior a una recreativa.

4

Mario Bros fue el primer gran éxito de Nintendo. Con más de veinte años de antigüedad sigue perdurando en las consolas actuales.



5

Con Zelda y su leyenda empezó una saga de RPGs que aún pervive.

6

Sonic de Sega fue y es competidor de Mario.

7

Fatal Fury 2 fue un claro exponente de la buena calidad que ofrecía la Super Nes de Nintendo.

NES (*Nintendo Entertainment System*), primera consola de videojuegos con tecnología de 8 bits, con unas posibilidades para gráficos y sonidos comparables a los microordenadores personales de la época.

Su verdadero triunfo llegó con el lanzamiento de la saga *Mario Bros*, juego de plataformas de concepción única, con gran colorido y música, con una jugabilidad increíble, ideal para toda la familia. En Estados Unidos causó furor e incluso se organizaban concursos para jugadores. Fue tal su fama que *Mario* fue uno de los primeros personajes en llegar al cine y televisión. Para acompañar el éxito de *Mario* apareció en escena *Zelda*, creación también del señor Miyamoto.

No tardó mucho en aparecer una gran competidora también procedente de Japón, *Sega*. Llegó a occidente y se unió al mundo de los 8 bits con su consola *Master System* y su personaje bandera: *Sonic*.

La saga *Sonic* compitió, y lo sigue actualmente, con su opuesto *Mario Bros*. Sin embargo, *Sonic* descubrió el poder de los juegos de plataformas con *scroll parallax* para consolas de 8 y 16 bits, con una velocidad y una suavidad de movimiento increíbles para la época.

Técnicamente Sega era mejor que Nintendo, y su juego *Sonic* poseía una endiablada velocidad, mejor sonido y un diseño de niveles asombroso.

La batalla estaba servida entre las dos compañías, que propiciaron un resurgimiento del sector lúdico para el hogar. Surgieron nuevos modelos con potencia de 16 bits, como la Super Nes de Nintendo y la Sega Mega Drive. En esta ocasión, Nintendo superó a Sega. La Super Nes proporcionaba unos gráficos más llamativos y sonidos más claros.

Nintendo pasó directamente a los 64 bits, comercializando la Nintendo 64 para competir con la *Sega Saturn*. Este avance de la tecnología acompañaba a juegos cada vez más vistosos, lo que daba lugar a licencias de las recreativas del momento sin apenas pérdida de calidad.

OTRO TIPO DE CONSOLAS

No podemos olvidar equipos como la prestigiosa *Neo-Geo* de SNK, que sin lugar a dudas tenía mejores prestaciones que sus homólogas de Nintendo o Sega, o la *Neo-Geo CD* con juegos en compact disc. Sin embargo, su elevado precio (debido a que sus juegos eran los mismos que la compañía programaba para las recreativas) fue el motivo de su escaso éxito comercial.

En la época de los 64 bits, surgió la poderosa compañía Sony, y volvió a traer el reinado a las consolas en los hogares, con la *PlayStation*. Sin duda, la consola que cuenta con el mayor número de juegos de la historia.

Paralelamente al éxito de las videoconsolas, los juegos también se disfrutaban en otro tipo de aparatos, los ordenadores personales. Pero esta es otra historia.

DEFINICIÓN

Un scroll es el efecto de realizar una panorámica o desplazamiento de la pantalla de juego con vista lateral. Cuando un juego posee varios planos de fondos movibles se denomina scroll parallax, y funciona simulando, por ejemplo, el efecto que produce mirar por la ventanilla de un tren en movimiento; los planos más lejanos se desplazan a menor velocidad que los más cercanos.

En el próximo número...

... hablaremos de los videojuegos para los primeros ordenadores de 8 y 16 bits que compartieron reparto con sus hermanos de las consolas.

Contenido CD-ROM 2

▶ AUDIO

■ RipCast Streaming Audio Ripper 1.3
Programa completo con el que podremos grabar audio de tipo streaming como MP3 separados.

■ FruityLoops 3.3.0
Podrás realizar efectos de sonido realmente espectaculares, mezclar y crear tus propias melodías de un modo muy profesional.

■ Rebirth 2.0
Todo un clásico mundial en cajas de ritmos virtuales para la realización de música de baile.

■ Logic 6

Programa realmente muy eficaz para la secuencia de pistas de audio a nivel profesional.

■ Reason 2
Uno de los programas de audio para ordenador mas innovadores de los últimos tiempos, un auténtico estudio virtual.

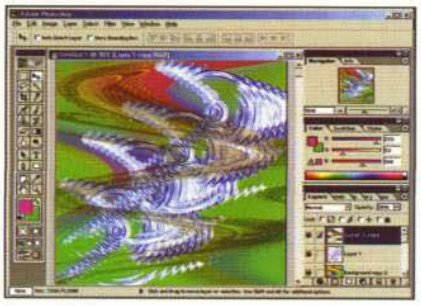
■ Samples variados
Algunos samples de muestra que puedes incluir en tu juego.

▶ DISEÑO 2D

■ AMI Graphic Workshop Pro 2.0a 39
Flexible editor para ver, convertir, procesar y optimizar imágenes.

■ Color Pilot 4.41
Excelente herramienta para corregir el color de imágenes capturadas digitalmente para así darles un aspecto más natural.

■ Adobe Photoshop 6



Sin duda, el rey de los programas de diseño gráfico. Podrás crear y retocar imágenes, y añadirles muchísimos filtros y efectos, todo ello con la mayor profesionalidad.

■ CBN Assistant 2
Un complemento ideal para el Photoshop. Te permite aplicar colores naturales a tus imágenes.

■ TraceArt 2.7d
Interesante aplicación con la que podrás convertir los bitmaps en imágenes vectoriales con una gran calidad.

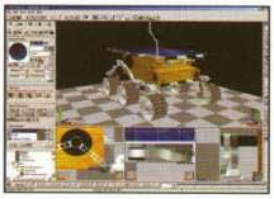
▶ DISEÑO 3D

■ 3D Photo Builder 1.1
Pequeño editor para crear imágenes 3D de un modo rápido, sencillo y ameno.

■ 3D Grapher 1.2
Genera superficies de dos funciones variables que pueden visualizarse desde cualquier perspectiva en tres dimensiones.

■ VR4MAX Navigator 1.20
Entorno de realidad virtual interactiva en tiempo real.

■ 3D Canvas 5.7



Para construir modelos 3D complejos utilizando primitivas, deformando y generando imágenes y animaciones profesionales.

■ Blender Creador 3d 2.23
Completa suite para creación de contenido 3D, con herramientas para modelación, animación, renderizado y post-producción.

▶ PROGRAMACIÓN

■ DJGPP 2.0
Librería multimedia inter-plataformas muy útil para manipular los gráficos y el audio de nuestro juego.

■ SDL 1.2.4
Buenísimas librerías que nos serán de gran ayuda. Originariamente escrito en C, funciona igualmente en C++ a nivel nativo.

■ Allegro 4.0.2
Librería gratuita para desarrollar juegos que además contiene funciones matemáticas, funciones 3D y otras interesantísimas prestaciones.

■ SGI OpenGL for Windows Software Development Kit (SDK)
Kit completo de desarrollo para transferir un carácter indiscutiblemente profesional al juego.

■ Microsoft DirectX 8.1 Software Development Kit
Otro excelente kit que nos permitirá mejorar nuestro juego hasta extremos insospechados.

▶ EMULADORES Y JUEGOS

■ Atari 2600

Todo un clásico. Incluye más de 500 juegos, entre ellos, Elk Attack, Okie Dokie, Oystron y This planet sucks. Para no aburrirse...

■ Emulador de Nintendo
La célebre consola emulada. Incluye los juegos: SuperMario Bros, Zelda, Bugs Bunny, Arkanoid, Galaxian, Kung-Fu, Conan, Batman, Donkey Kong, etc...

■ Otros juegos
Podéis encontrar además dos divertidos juegos: LaserAge 2.0, excelente arcade ideal para los fans de Galaxia, y Spider-Man, demo del magnífico juego 3D.

▶ VÍDEO

■ VAMP Media Center 3.11

Herramienta de audio y vídeo para realizar pequeños clips y otras aplicaciones multimedia. Es además *peer-to-peer*.

■ Canopus ProCoder Demo 1.0
Potente conversor de formatos de vídeo, para crear vídeos digitales de una gran calidad.

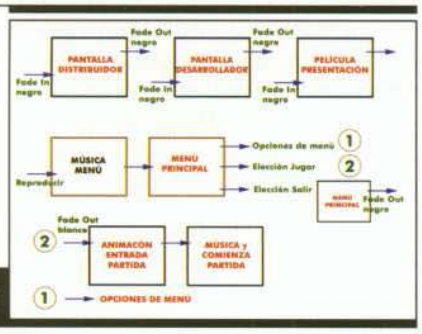
Diseño de los elementos I: desarrollar la idea

En el número anterior definimos, de una forma práctica, todos los aspectos de nuestro juego. Elegimos un tipo de juego de acción en 3D con varias vistas de cámaras seleccionables por el jugador, un estilo gráfico algo futurista y una ambientación sonora con bastante ritmo. El juego tendrá posibilidad multijugador y la jugabilidad se centrará en el combate con otros adversarios en un terreno acotado. Pues ya podemos ponernos manos a la obra.

(■) EL GUIÓN

Básicamente hay dos tipos de guiones: el literario y el técnico. Por ejemplo, una aventura gráfica precisa de los dos: el técnico para describir las facetas gráficas, sonoras o de programación, y el literario para contar el desarrollo de la aventura (descripción de personajes, diálogos o situaciones). Los guiones varían según el tipo de juego y para explicar cada una de estas variaciones necesitaríamos muchas páginas. Por ello, nos centraremos en el tipo de juego que hemos elegido. Generalmente, en un videojuego de nuestras características –donde la acción se centra en el combate entre jugadores y en un único tipo de situación– no existen diálogos entre personajes ni cambios para seguir una historia. Básicamente, esta aventura la fabrica el mismo jugador con su actuación o elección en el menú principal. Precisamente, de él depende si se combate en un terreno arenoso o en un medio acuático, o si se compite con otro jugador humano o con el ordenador. Por lo tanto, realizaremos un guión técnico.

Este guión servirá para conducirnos a través del desarrollo y, fundamentalmente, se trata de una especie de cuaderno de notas en el que describiremos todas las facetas del juego de manera secuencial. Es decir, en qué orden y qué elementos aparecen en nuestro juego desde que empieza su ejecución. Podemos ayudarnos con métodos como, por ejemplo, un esquema gráfico como se muestra en la figura 1, donde observamos nuestro guión esquematizado, describiendo cada secuencia técnica dentro de cuadros.



1

En la figura se muestra un esquema gráfico del guión de nuestro juego.

(■) GUIÓN TÉCNICO DE *ZONE OF FIGHTERS*

Comenzamos el guión describiendo paso a paso la ejecución del juego:

- **1.** Aparece pantalla de presentación del distribuidor desde fondo negro (FADE IN –fundido de entrada– a negro)
- **2.** Desaparece pantalla del distribuidor hacia fondo negro (FADE OUT –fundido de salida– a negro)
- **3.** Aparece pantalla del desarrollador desde fondo negro (FADE IN –fundido de entrada– a negro)
- **4.** Desaparece pantalla del desarrollador hacia fondo negro (FADE OUT –fundido de salida– a negro)
- **5.** Aparece película de presentación (FADE IN negro)
- **6.** Termina película de presentación
- **7.** Inmediatamente activamos música del menú principal
- **8.** Inmediatamente aparece menú principal
- **9.** Esperar elección del jugador
- **10.** En caso de elección del jugador cambiar de menú desplegando las opciones
- **11.** En caso de elección de salida del juego FADE OUT a negro de la pantalla del menú
- **12.** En caso de comenzar partida; FADE OUT a blanco de la pantalla del menú principal
- **13.** Comienza la animación de entrada a una nueva partida
- **14.** Activar música de juego
- **15.** Comienza partida
- **16.** Si el jugador elige opciones durante la partida se para la acción y aparece menú de opciones.
- **17.** Si el jugador decide salir de la partida, FADE OUT a blanco y aparece menú principal con FADE IN desde blanco.
- **18.** Si muere el jugador, termina la partida. Aparece pantalla o acción de GAME OVER y FADE OUT a blanco. Aparece el menú principal con FADE IN desde blanco.

NOTA

Lo importante es que los encargados de interpretar el guión deben entenderlo fácilmente para poder desarrollar correctamente el diseño.

Esto es básicamente una forma de realizar un guión técnico.

BOCETOS Y *STORY BOARD*

Los bocetos representan la antesala del aspecto del juego. Son dibujos, más o menos artísticos, que plasman la idea que se tiene de los gráficos. Generalmente, se suele dibujar a los protagonistas, los decorados y sus elementos como construcciones, vegetación, etc. Sin embargo, siempre es muy conveniente tener al menos algunos trazos de otros aspectos como los menús, iconos, indicadores de pantallas y cosas así.

No vamos a enseñar en esta obra a dibujar bocetos, pero sí daremos una orientación de cómo deben ser. No es importante que el dibujo sea una auténtica obra de arte, pero sí que la idea que se quiere transmitir sea reconocida perfectamente por los grafistas. Los bocetos suelen estar en blanco y negro cuando se realizan para el modelado posterior. El uso del color será importante y necesario cuando el diseñador, por ejemplo, describe un tipo de ambiente especial y es imprescindible reflejar el colorido. También sería útil el color, por no decir fundamental, para el grafista 2D que realiza las texturas de los modelos; es primordial que éste sepa si el casco de la nave es gris metálico, si la piel del personaje principal es oscura o si el terreno es de tierra roja o amarilla para poder realizar su trabajo.

2

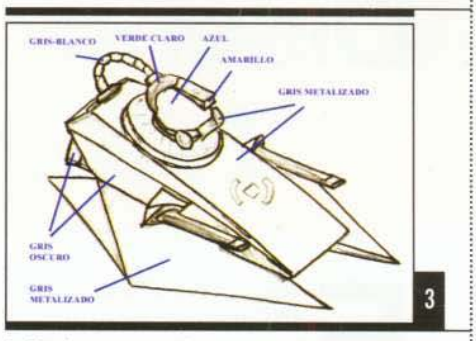
Ejemplo de boceto coloreado, aunque los bocetos suelen ser en blanco y negro.

No obstante, es suficiente con indicar los colores por escrito sobre el boceto en blanco y negro, ya que la misión de estos dibujos es servir de guía para el modelador y el texturizador.

Otra regla básica a tener en cuenta es desde qué ángulo se ve el dibujo en el papel. Todo depende de qué queremos representar. Es conveniente dibujar a los protagonistas del juego en un mínimo de tres vistas, en las que se puedan distinguir la parte frontal, lateral y trasera del modelo. Si lo que queremos es dibujar algún tipo de construcción, de formas más o menos geométricas, basta con una perspectiva isométrica en la que se vean al menos dos caras. Para ambientes y elementos de decoración (piedras, vegetación, etc.) es suficiente con una vista lateral.

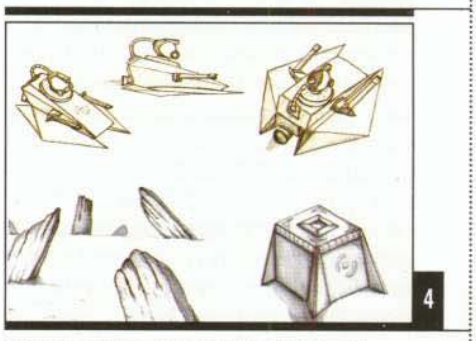
El *storyboard* es similar a un guión técnico esquematizado, pero con la salvedad de que incluye viñetas dibujadas con anotaciones de texto. Se utiliza para ilustrar las fases que tendrá un anuncio, una película o, en nuestro caso, una animación. Las viñetas contienen el desarrollo de una y el desarrollo de ésta se refleja con textos técnicos que indican movimientos de cámara o de modelos e incluso cambios del entorno. El *storyboard* ayuda a los grafistas en el modelado y la animación.

En *Zone of Fighters* tendremos una animación al comienzo de cada nueva partida, consistente en la entrada de las bionaves en el interior de las urnas que contienen los terrenos de combate. En la figura 5 podemos ver el *story board* completo de la animación. En este caso, el sistema escogido es completamente gráfico. Cada viñeta se divide en dos partes: el dibujo de la izquierda, más detallado, representa cómo empezaría la acción y el de la derecha, más pequeño y esquemático, el desarrollo de esta acción mediante flechas.



3

Indicaciones por escrito del color sobre un boceto en blanco y negro.



4

Ejemplos de bocetos: es conveniente tener varias vistas de cada modelo.

LOS PROTAGONISTAS DEL JUEGO

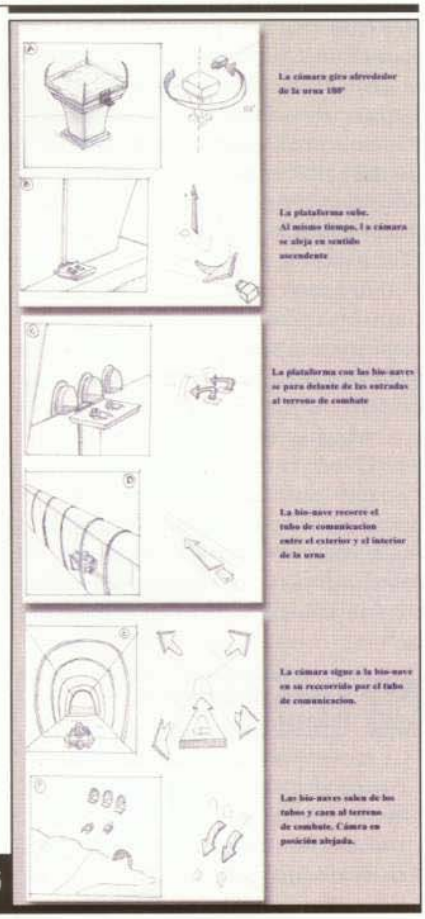
Entendemos por "protagonistas" a todos los personajes que intervienen en el juego. El papel del jugador lo interpretará el protagonista principal, ya sea un guerrero con espada, un jugador de fútbol, etc. De todas formas, en muchos casos, nuestro protagonista puede quedar oculto en el interior de un avión comercial o, en nuestro caso, de una nave de combate. Los demás protagonistas serán los que, de alguna u otra manera, intervienen en la historia del juego, bien como enemigos directos en la acción o como extras. Básicamente, estos personajes dan sentido a la historia del juego, por lo que no hace falta comentar que un diseño correcto es fundamental para dar credibilidad a la historia.

PERSONAJE PRINCIPAL

Más del 50 por ciento del éxito de un juego se debe al protago-

nista principal. Es con quien toma contacto y se identifica el jugador en todo momento. Un claro ejemplo de esta importancia la encontramos en los RPGs (ROL). En este tipo de juegos es trascendental un sistema de configuración del protagonista principal por parte del jugador, ya que el desarrollo de la historia del juego depende en gran medida de este hecho. Actualmente, raro es el juego que no permita una configuración personal del protagonista principal; desde asignarle el nombre del jugador hasta cambiarle el aspecto físico.

Con la llegada de los FPS o juegos en primera persona el jugador se aproxima al máximo al personaje principal, ya que se convierte en él mismo y ve a través de sus ojos, debido a la perspectiva subjetiva que se tiene de la acción. La llegada del sistema de cámara en tercera persona –en parte originado por la necesidad que el jugador demanda de verse a sí mismo representado por un héroe– contribuyó a mejorar la calidad del diseño de este elemento esencial en todo videojuego. Aunque, generalmente, es posible cambiar de vista en un juego en 3D, es la tercera persona, sin duda, la preferida por los jugadores; tanto que incluso los FPS han adoptado esta opción. Si hacemos una pequeña observación, el porcentaje de usuarios de juegos de acción de sexo femenino aumentó considerablemente tras la salida al mercado del juego *Tomb Raider*, sólo por el hecho de que la protagonista principal era una mujer.



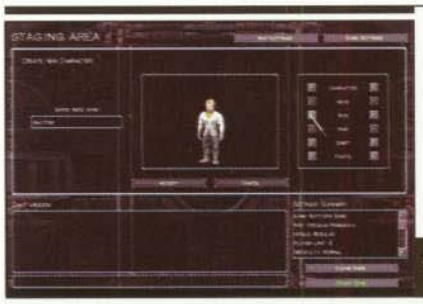
5

Ejemplo de Story Board completo de la animación.

Cada vez son más los tipos de acciones que el personaje principal puede realizar, con una calidad de movimientos pasmosa y una apariencia aún más real. Este aspecto del diseño es importantísimo para asegurar el éxito de un juego. Pensemos un momento por qué triunfó mundialmente el juego *Prince of Persia* entre los primeros juegos para ordenador. Aparte de tener una ambientación gráfica estupenda, el personaje principal estaba dotado de unos movimientos asombrosos. A veces el desarrollo del juego se hacía pesado y monótono, pero el simple hecho de ver cómo tu personaje se movía era suficiente para tenerte horas delante de la pantalla.

Puede ocurrir que, aunque nuestro protagonista sea humano, va conduciendo algún tipo de vehículo; por ello, el jugador sólo ve dicho artefacto. Entonces el diseño del protagonista principal se centra en el vehículo y no en el ser antropomorfo.

En *Zone of Fighters* nuestro protagonista maneja una bionave de combate; si observamos en la figura 2 el diseño de este vehículo, vemos cómo parte de nuestro personaje asoma de la nave; por lo tanto, hemos conseguido una representación más cercana del jugador.



6

La configuración del protagonista principal ayuda más a que el jugador se identifique con él.

OTROS PERSONAJES

A este sector pertenecen todas las demás criaturas o artefactos que comparten historia con el personaje principal. Realmente son los que dan la razón de ser al juego. Sin ellos no tendríamos acción, ni siquiera una historia que jugar. Hay muchas clases de estos personajes, dependiendo de su papel. Los tenemos protagonizando la historia codo con codo con nuestro intérprete o bien de segundones en papeles de relleno. Pero, sin lugar a dudas, son los elementos más difíciles y gratificantes de programar en un juego. Además de un diseño gráfico, hay que sumarle un diseño de

NOTA

Los personajes deben ser capaces de mostrar cierta inteligencia, para que así el juego sea más interesante.

7

Price of Persia fue de los primeros juegos en dotar a los personajes de movimientos realistas.

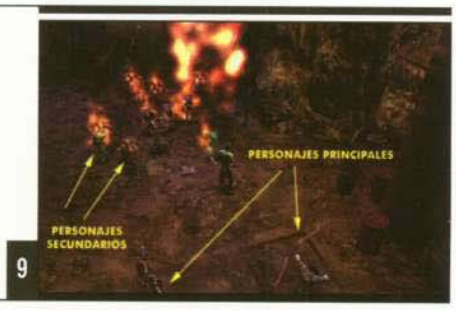
8

En los juegos de estrategias, la Inteligencia Artificial de los personajes que controla el ordenador posee un nivel muy elevado de realismo.

comportamiento. Implementar cierta inteligencia a estos personajes constituye, hoy día, uno de los pilares tecnológicos más importantes en el desarrollo de videojuegos. Si no, que se lo digan a Iwatani, el creador del *Pacman*, que la mitad del tiempo que duró su desarrollo lo gastó en diseñar el comportamiento de los fantasmas del juego, que fue –sin lugar a dudas– lo que mantuvo pegado al juego a millones de personas.

Toma mayor importancia, en los días que corren, la IA (Inteligencia Artificial) de los personajes secundarios, debido a la jugabilidad que demanda el jugador, cada vez más exigente. Un ejemplo bien claro de lo imprescindible que se vuelve el diseño de la IA de personajes lo encontramos en los juegos de estrategia, donde la jugabilidad sólo existe si las unidades enemigas dan la sensación de estar comandadas por otro jugador. El ansia de superación invita al usuario a seguir jugando y, para que eso ocurra, el comportamiento de los enemigos tiene que parecer imprevisible y con cierto sentido de competición. Los comportamientos fijos y, por consiguiente, predecibles, provocan aburrimiento en la acción.

El diseño de los personajes secundarios depende, en gran medida, del tipo de juego al que pertenecen. En juegos tipo Arena o juegos de combate como *Quake III* o *Unreal Tournament*, ocurre que los personajes secundarios pueden ser otros jugadores conectados a la partida por cable o Internet. *Zone of Fighters* es un juego de este tipo, pero con el añadido de otros personajes que viven en el campo de batalla y que, forzosamente, forman parte de la acción entre los jugadores. Así, nos encontramos con animales y plantas carnívoras letales que ayudan a añadir aún más emoción a la partida. La naturaleza inesperada de la situación de estas criaturas en el terreno de combate puede dar una sensación de comportamiento imprevisible, aunque en realidad no lo sea, debido a que la atención del jugador está desviada por el verdadero núcleo de la acción, el combate entre las bio-naves.



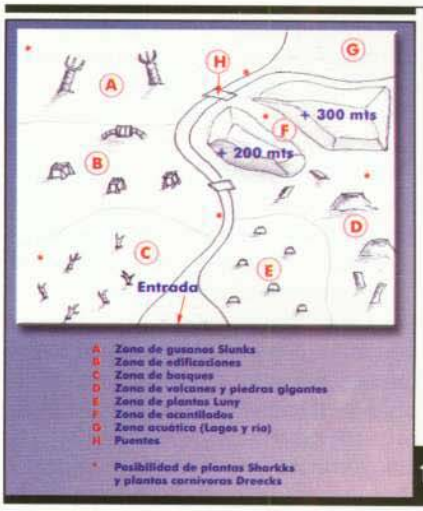
9

El comportamiento de los personajes secundarios de un juego tiene que parecer lo más real posible para ganar en jugabilidad.

EL MAPEADO: LAS ZONAS DE COMBATE

Todo juego transcurre en un entorno definido. Podemos viajar por el espacio, recorrer una tierra imaginaria o luchar en el interior de cavernas, pero todas estas situaciones deben estar previamente diseñadas. Al igual que se dibuja el mapa de un país o una región, es necesario hacer lo mismo en nuestro juego.

En el mapeado, además de dibujar las distintas zonas, niveles o habitaciones por donde transcurrirá el juego, hay que indicar qué decoración, seres o vegetación se pueden encontrar, así como todos aquellos objetos especiales que pueda hallar el jugador como armas, munición, llaves, bonos, etc.



10

Boceto del mapeado del primer nivel del juego Zone of Fighters.

Como ya sabemos, *Zone of Fighters* transcurre en grandes superficies de terreno con bosques, ríos, volcanes, edificios, animales, etc. Igualmente, habrá repartidos distintos objetos que pueden recoger los jugadores. Entonces, para que el grafista y el programador sepan cómo fabricar y colocar todos estos elementos, es necesario hacer un croquis o dibujo de los terrenos indicando el lugar exacto donde se ubicarán.

Hay casos en que un objeto o animal pueden aparecer en cualquier parte del mapeado aleatoriamente; por ejemplo, los bonos de puntos y la munición o el ataque de un gusano. En ese caso, basta con indicarlo o marcar la zona de la posible situación. En la figura 9, podemos ver el mapeado de la primera zona de combate de nuestro juego con una leyenda explicativa.

En el próximo número...

... definiremos nuestra bionave de combate: aspecto, acciones, armas, etc., así como todos los demás personajes del juego. Estudiaremos sus comportamientos y diseñaremos su forma.

Modelado en baja poligonización con Milkshape3D (I)

Modelar en baja poligonización consiste en construir modelos en 3D con pocos polígonos. El polígono más pequeño que podemos formar es el triángulo. Éste, a su vez, está formado por tres vértices unidos entre sí. Este sistema es básico para el modelado de objetos para videojuegos, ya que de la cantidad de polígonos que estos tengan dependerá el rendimiento del juego.

Para modelar en baja podemos utilizar varios procedimientos. El más sencillo es crear objetos por medio de primitivas, como cajas, cilindros o esferas, editando luego sus vértices. Sin embargo, aunque este sistema no permite modelar objetos complejos -como formas orgánicas- sin utilizar gran cantidad de polígonos, es muy útil para fabricar otros más sencillos como edificios, vehículos, etc. El siguiente sistema es más laborioso, pero también más flexible. Permite modelar cualquier tipo de objeto ya sea orgánico o inorgánico y consiste en ir formando las caras del objeto utilizando triángulos, los cuales creamos previamente uniendo vértices.

EMPEZAR CON MILKSHAPE 3D

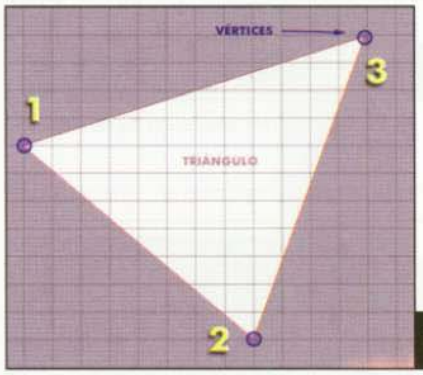
Milkshape3D surgió de la mano de los creadores del *HalfLife* para modelar los objetos que luego utilizarían en su juego. Posteriormente, se abrió al publico como aplicación sharewareare y, debido a su coste más que asequible, es el más usado por los grafistas *freelance*. Es fácil de usar y soporta todos los formatos conocidos para juegos. Es tremendamente popular en la comunidad de desarrolladores de *Blitz 3D*, tanto que incluso existe una utilidad gratuita que se instala en el *Milkshape 3D* (plugin) para exportar los modelos al formato gráfico del *Blitz 3D* (*.B3D*). *Milkshape 3D* es básicamente un modelador de baja poligonización que además permite crear esqueletos y dotar de animación a los modelos. Aun así, nosotros nos centraremos sólo en el modelado, ya que la animación de nuestros objetos la realizaremos con otra aplicación más especializada.

Al ejecutar el programa, podemos ver que está dividido en distintas zonas. El área de vistas, la zona de edición, la ventana de mensajes y el Keyframer para la animación.

ÁREAS DE VISTAS

Esta zona es similar a cualquier otra aplicación 3D. Tenemos cuatro ventanas, las cuales muestran una vista distinta del objeto (vista lateral, frontal, trasera y perspectiva). En "Viewports" situado en el menú "Windows", podemos cambiar la disposición y el número de estas vistas. Pulsando el botón derecho del ratón sobre cualquiera de las vistas, abriremos un menú con todas las opciones de visualizado, que desde este momento llamaremos "Menú Emergente". Eligiendo la opción "Maximize", la ventana de visión sobre la que estamos ocupará toda la pantalla. Para volver a la configuración de vistas anterior, basta con elegir de nuevo "Maximize".

Antes de continuar, vamos a ocultar el Keyframer y la ventana de mensajes, ya que no los vamos a utilizar. Para ello, deseleccionamos en el menú



1

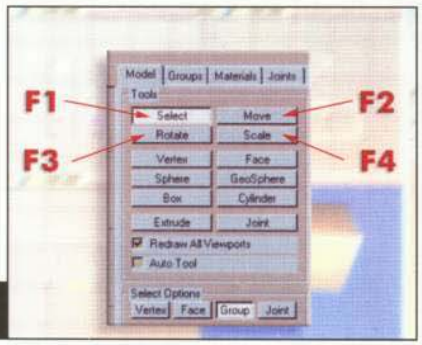
Podemos modelar los objetos usando triángulos, los cuales creamos uniendo vértices.

2

En las ventanas de vistas se nos muestran distintas perspectivas del objeto.

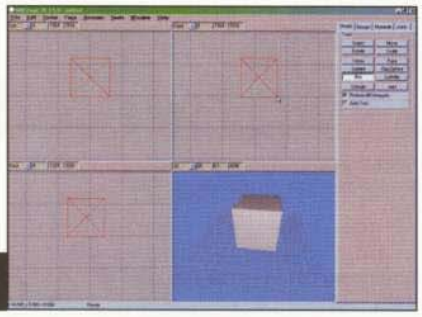
NOTA

Para acceder al menú emergente de opciones, es suficiente con pulsar el botón derecho del ratón sobre cualquiera de las vistas.



3

En la zona de edición están todas las herramientas necesarias.



4

Creamos un cubo arrastrando el ratón. El color rojo indica que ya es un objeto.

NOTA

En cada vista, la línea de color amarillo representa el eje X, la azul el Y y la rosa el eje Z.

NOTA

Un vértice, cara u objeto, se vuelve de color rojo cuando está seleccionado y blanco cuando no lo está.

"Window" las opciones "Show Keyframer" y "Show Message Window". Observamos que en la parte superior de cada vista hay un encabezamiento (caption) con cuatro casillas. La primera es un pequeño menú desplegable que permite cambiar la vista en esa ventana, la segunda casilla nos indica el tamaño del *grid* o parrilla modificando el zoom de la cámara, y la tercera y cuarta casilla indican respectivamente lo cerca y lejos que puede llegar el campo de visión de la cámara. Nosotros vamos a utilizar los valores estándar en las vistas lateral, frontal y trasera, (4, -1024, 1024) y para la vista 3D cambiaremos a 50, 0.1 y 4096.

Los encabezamientos se pueden ocultar deseleccionando en el menú "Window" la opción "Show Viewport Caption".

Observamos también que en el centro de cada ventana de vistas hay tres líneas de colores. Estas líneas representan los ejes X (de color amarillo), Y (de color azul) y Z (de color rosa). Estos ejes se pueden ocultar deseleccionando "Show Axis" en el "Menú Emergente". A continuación vamos a situarnos en la vista 3D. Para poder rotar la cámara, movemos el ratón manteniendo pulsado el botón izquierdo. Para desplazarla sobre la vista, haremos lo mismo con el ratón, pero pulsando la tecla *CTRL*, y para acercar o alejar la cámara dejaremos pulsado el botón izquierdo más la tecla *SHIFT* pero moviendo el ratón hacia arriba o abajo.

Todos estos procedimientos son aplicables al resto de las ventanas de vistas, a excepción del zoom de cámara que sólo es posible si está activada la opción "Move", la cual explicaremos más adelante.

ZONA DE EDICIÓN

Es aquí donde se encuentran todas las herramientas necesarias para trabajar con *Milkshape 3D*. La mayoría de las funciones más utilizadas para modelar y editar los objetos van asociadas a una tecla de función. Es muy conveniente tener esto bien claro para trabajar mucho más cómodamente. La zona de edición está compuesta por cuatro grupos de funciones: "Model" (modelado), "Groups"(grupos), "Materials"(materiales) y "Joints"(uniones).

En "Model" tenemos todas las herramientas necesarias para construir y modificar los objetos 3D, desde simples vértices hasta primitivas completas. En "Groups" encontramos lo necesario para seleccionar y agrupar objetos de un modelo. En "Materials" podemos cargar, modificar y asignar los materiales que luego irán en cada objeto. Y por último, en "Joints" se engloba todo lo necesario para crear esqueletos mediante la unión de huesos.

CREAR, SELECCIONAR UNA PRIMITIVA

Para familiarizarnos con las opciones de edición básicas del *Milkshape 3D*, vamos a utilizar primitivas.

En primer lugar, vamos a crear un cubo en medio de la cuadrícula. Seleccionamos la pestaña "Model" de la zona de edición. Pulsamos sobre el botón "Box". A continuación, nos colocamos en la vista frontal (front) y sin dejar de pulsar el botón izquierdo del ratón nos desplazamos. Observamos cómo se va formando un cubo en todas las vistas. Al soltar el botón, el cubo se vuelve de color rojo indicando que ya es un nuevo objeto. Si volvemos a pulsar el botón y desplazamos el ratón, estaremos creando otro cubo nuevo y el primero se vuelve de color blanco. (ver fig. 4).

Como pequeño ejercicio y aprovechando que hay un objeto como referencia, podéis practicar el desplazamiento de la cámara en la vista 3D. Cuando hayáis practicado suficiente probad en las demás vistas. Si en algún momento perdéis el panorama el cubo, se

puede recuperar la vista original eligiendo la opción "Reset View" en el "Menú Emergente".

A continuación aprenderemos a seleccionar las partes del cubo que hemos creado. En la parte inferior de la zona de edición están las "Select Options" (opciones de selección). Podemos seleccionar vértices, caras o polígonos, grupos u objetos y uniones de huesos. Elegimos la opción "Select" (F1) en "Tools" y pulsamos en el botón "Vertex" para seleccionar los vértices inferiores del cubo. Nos situamos en la vista frontal y hacemos *clic* desplazando el ratón para abrir un área de selección que englobe los dos vértices inferiores. Una vez seleccionados se vuelven de color rojo. Observamos en la vista izquierda (Left) cómo se ha seleccionado sólo un vértice. (ver fig 5). Esto significa que al seleccionar en la vista frontal hemos elegido sólo los vértices situados delante; para elegir también los vértices traseros (los que no se ven) debemos deseleccionar la casilla "Ignore Backfaces" (ignorar caras traseras)

Para seleccionar una cara o polígono se realiza la misma operación, pero eligiendo la opción "Face". Hay que tener en cuenta que para seleccionar una cara es necesario elegir al menos dos vértices. Para elegir un objeto o grupo de objetos basta con pinchar sobre ellos con la opción "Group" elegida.

MOVER, ROTAR Y ESCALAR UNA PRIMITIVA

Vamos a mover nuestro cubo por la cuadrícula. Primero lo seleccionamos y luego pulsamos el botón "Move" (F2). Ahora es cuestión de hacer *clic* en cualquier punto de la vista deseada y desplazar el ratón con el botón izquierdo pulsado. Cuando elegimos ciertas herramientas como mover, rotar o simplemente crear un cilindro, aparecen en la parte inferior de la zona de herramientas los diálogos de opciones correspondientes.

Sabiendo esto, podemos también desplazar de una manera más exacta un objeto, y es introduciendo valores para los respectivos ejes y pulsando el botón "Move" de "Move Options". (ver. Fig 6).

Para rotar (F3), tenemos que tener en cuenta el punto de referencia donde girará el objeto. Para elegir este punto tenemos tres opciones: "Center of Mass" (centro del objeto), "Origin" (origen del objeto) y "User Point" (Punto por el usuario). Con la primera opción, se rota a partir del centro del objeto; con la segunda, desde su origen (determinada por la situación de los ejes); y con la tercera, a partir de donde hagas *clic* con el ratón.

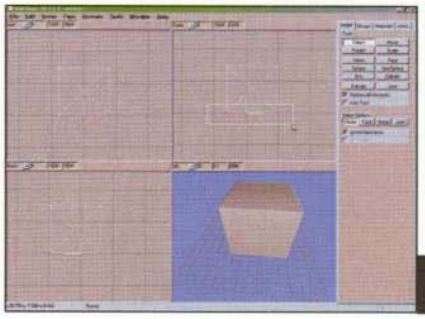
Elijamos la opción "Center of Mass". Hacemos *clic* en la vista frontal y desplazamos el ratón con el botón izquierdo pulsado. Vemos cómo el objeto gira sobre su centro. Si pinchamos en la opción "Origin", el punto de rotación se colocará en el centro de los ejes X, Y y Z y el cubo pivotará sobre ellos. Podemos rotar con valores prefijados en las casillas numéricas situadas en "Move Options".

El procedimiento para cambiar de tamaño el cubo es similar a los anteriores. Pulsando el botón "Scale" (F4) podremos escalar el objeto a partir del punto de origen que marquemos en "Scale Options". (ver fig. 7).

Todas estas funciones de edición funcionan igualmente con cualquier selección que hagamos, ya sean vértices, caras u objetos. Es conveniente que practiques con estas opciones de edición y adquieras soltura en su manejo. Prueba con distintas primitivas, vértices o caras.

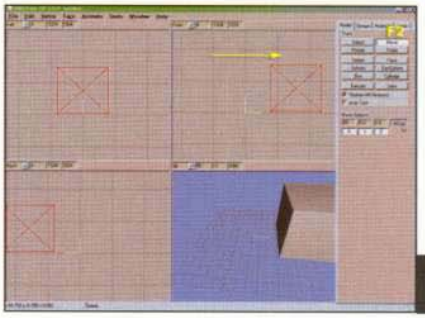
En el próximo número...

... aprenderemos todo lo necesario para empezar a modelar los objetos de *Zone of Fighters*.



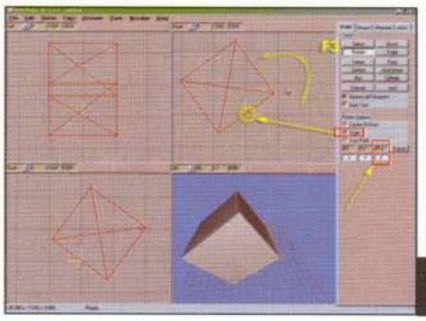
5

Selección de un vértice en la vista izquierda.



6

Pulsando el botón "Move", desplazamos el objeto.



7

Escalamos el objeto con el botón "Scale".

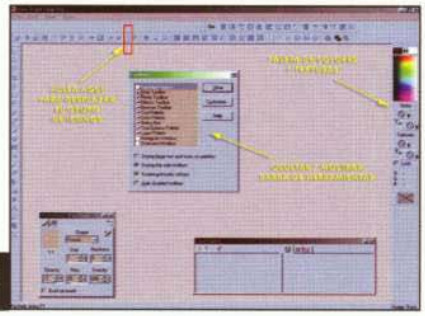
NOTA

La opción "Ignore Backfaces" permite seleccionar o no los vértices o caras que no se ven en una vista.

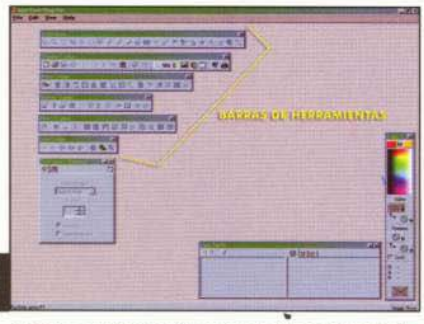
NOTA

Milkshape3D tiene hasta diez niveles de *undo* (deshacer) (CTRL + Z).

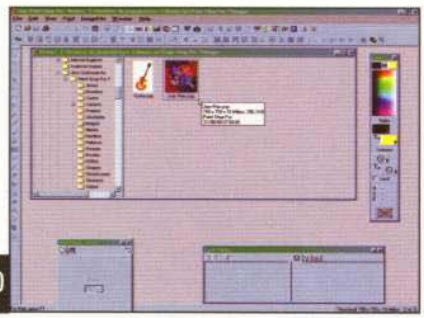
MODELADO 2D



8 Vista del entorno de Paint Shop Pro.



9 Las barras de herramientas se desplazan libremente.



10 Empezando a trabajar: abrimos una imagen.

¿QUÉ ES *PAINT SHOP PRO* Y PARA QUÉ LO UTILIZAREMOS?

Paint Shop Pro es una aplicación para crear, editar y retocar imágenes. Gracias a su amigable interfaz, su precio y sus capacidades, se ha convertido en una buena alternativa a programas más populares, como *PhotoShop*.

Es ideal para dibujar todos los gráficos 2D que nuestro juego necesita como: pantallas de presentación, logotipos, iconos y bitmaps para sprites y texturas. Puede manejar cualquier tipo de formato de imagen conocido, trabajar con múltiples capas, crear y editar vectores y permite utilizar varios niveles de undo (hacer y deshacer acciones.)

PRIMERA TOMA DE CONTACTO

La filosofía de diseño de este programa se asemeja a la utilizada por Adobe *PhotoShop*, así que, si estás familiarizado con la aplicación de Adobe, no tendrás muchos problemas en habituarte a él.

Al ejecutar *Paint Shop Pro* podemos observar varias paletas esparcidas por la pantalla y varias barras de herramientas, las cuales representan, mediante iconos, todas las opciones posibles del menú principal. Las barras de herramientas se pueden desplazar y colocar en el lugar de la pantalla que desees con el ratón; para ello, debemos mantener pulsado el botón izquierdo sobre las dos barras paralelas que se hallan en el comienzo de cada grupo de iconos mientras las desplazamos. (ver fig. 9). En la opción "ToolBars" del menú "View" podemos ocultar o mostrar estas barras de herramientas, así como las paletas. También pueden enrollarse y desenrollarse automáticamente, conmutando la acción en el botón de la flecha situado en la etiqueta.

Todas las opciones de cada herramienta se encuentran en la ventana "Tool Options".

EMPEZAR A TRABAJAR

Para comenzar a trabajar con *Paint Shop Pro* necesitamos tener una imagen. Podemos crearla nueva, cargar un archivo almacenado o importarla directamente desde un escáner o cámara digital. Para cargar una imagen almacenada elegimos la opción "Open" del menú "File"o pulsamos *CTRL + O*. Hay otra manera de seleccionar una imagen para cargarla y es utilizando la herramienta "Browse" pulsando *CTRL + B* o eligiendo la opción "Browse" en "File". Esta opción nos permite ver una miniatura de todas las imágenes que se encuentran en un directorio.

Basta con pulsar dos veces sobre la miniatura para cargarla directamente en el programa. Si lo que queremos es crear una imagen nueva, elegiremos "New" (CTRL + N). Posteriormente, aparecerá un cuadro de diálogo donde debemos elegir el tamaño, calidad y color del fondo. Una vez pulsado "OK" aparecerá una ventana nueva con el color que elegimos como fondo.

NOTA

El formato de imagen nativo del *Paint Shop Pro* tiene la extensión .PSP

TRUCO

Para ocultar o mostrar todas las paletas activas, pulsar la tecla de tabulación (TAB).Todas las opciones de cada herramienta se encuentran en la ventana "Tool Options".

En el próximo número...

... conoceremos las herramientas y nos prepararemos para empezar a realizar el logotipo de nuestro juego. Sólo con práctica, mucha práctica, dominarás *Paint Shop Pro 7*.

Herramientas para desarrollar el sonido de un juego

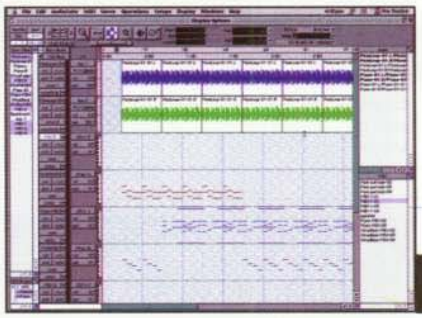
Antes de existir el sistema operativo Windows y los sistemas multimedia, la capacidad sonora en un PC de serie se limitaba a su altavoz interno. Lo más que podía generar un juego, por aquella época, era un montón de pitidos que, a veces, resultaban hasta molestos. Otros microordenadores habían conseguido con éxito reproducir sonido con cierta calidad, como en el caso del Amiga de Commodore que, prácticamente, basaba su publicidad en sus posibilidades multimedia.

Esta capacidad sonora no comenzó realmente en el PC hasta la llegada de las tarjetas de sonido, que proporcionaban al ordenador una expansión de su hadware de la que carecía. Los juegos empezaron a mostrar efectos especiales de sonido algo complejos y hasta música al mismo tiempo. El uso del PC para generar música sufrió un gran auge que, evidentemente, se extendió al sector lúdico. La necesidad de cubrir estas nuevas posibilidades provocó el nacimiento de numerosos softwares para la creación de música.

¿QUÉ PROGRAMAS SE UTILIZAN PARA HACER LA MÚSICA Y LOS EFECTOS DE SONIDO?

Por aquel entonces, antes de que aparecieran las tarjetas de sonido, sólo existían para PC interfaces MIDI para conectar el ordenador a instrumentos musicales. El ordenador proporcionaba, en este caso, secuenciadores que controlaban los eventos MIDI de los instrumentos. El más conocido fue el *CakeWalk*, que podía manejar hasta 256 pistas de eventos musicales simultáneamente. Las posibilidades musicales, en el ámbito profesional de aquellos tiempos, estaban en manos de los sistemas Apple y Atari. Los Macintosh de Apple acapararon toda la producción profesional de aplicaciones musicales con programas como *Perfomer*, *Pro Tools*, *MasterTrack* o *Visión*. Pero, para el usuario de a pie, estos sistemas eran económicamente inalcanzables, dando paso a los asequibles ordenadores Atari con interfaz MIDI incorporado y con aplicaciones realmente potentes como *Cubase*. Este programa se implementó posteriormente en los PCs, perpetuando así su existencia después de la desaparición de Atari y, hoy día, sigue siendo un estándar mundial en secuenciadores de mediano y alto nivel. Actualmente, con la versión 5 del *Cubase VST*, es posible manipular tanto MIDI como pistas de audio; forma parte, junto con otras aplicaciones como *Logic* o *Pro Tools*, del conjunto de herramientas más utilizadas para el empleo de instrumentos electrónicos en la creación de música para videojuegos.

Asimismo, el avance tecnológico en la generación de audio por ordenador y los nuevos formatos de audio han contribuido, en gran medida, al cambio en la utilización de sistemas para hacer música. Se han desarrollado aplicaciones para PC que emulan a la perfección cualquier generador de sonido externo como sintetizadores o cajas de ritmos. Ahora, por mucho menos dinero, es posible tener en tu ordenador más



1

El programa ProTools fue el más popular entre los profesionales que usaban ordenadores Macintosh de Apple.

2

El Atari 1040 ST introdujo el uso del ordenador para la música en los hogares gracias a su precio y su interfaz MIDI incorporado.

3

Steinberg introdujo sus secuenciadores CUBASE en los Atari para uso amateur y profesional, y actualmente es uno de los más populares.

4

El programa LOGIC de Emagic es muy eficaz para la secuencia de pistas de audio a nivel profesional.

5

El programa Rebirth es un clásico mundial en cajas de ritmos virtuales para la realización de música de baile.

6

A diferencia de otros editores de audio, Cool Edit Pro ofrece la posibilidad de mezclar pistas de audio como lo haría un secuenciador.

DEFINICIÓN

▸ ¿QUÉ ES UN SAMPLER?

Samplers o muestreadores digitales son aparatos que permiten capturar con bastante exactitud cualquier tipo de sonido externo y convertirlo posteriormente en audio digital (sampler o muestra) para poder ser procesado.

prestaciones que instrumentos electrónicos de elevado precio. Igualmente, la posibilidad de introducir audio a través de la tarjeta de sonido desde fuentes externas como micrófonos o instrumentos electrónicos ha dado lugar a la utilización del ordenador como *sampler*.

Uniendo a esto la velocidad de procesamiento y la capacidad de almacenamiento de datos que proporciona un ordenador, tenemos un cóctel explosivo del que beben millones de usuarios profesionales y *amateurs*.

Aparecen en escena infinidad de aplicaciones para generar música y efectos de audio de manera cómoda y potente como: *Rebirth* o *Fruity Loops* ampliamente utilizados en la creación de música electrónica de baile, sintetizadores virtuales, procesadores de efectos, etc.

La posibilidad de almacenar sonido digital originó la aparición de sistemas para manipular esta información de audio. Ahora es muy común grabar en el ordenador (como si de una grabadora se tratase) audio procedente de cualquier fuente de sonido y poderlo editar y transformar posteriormente. Este tipo de operaciones se realiza con aplicaciones denominadas "editores de audio". Existe un gran número de estos programas, pero todos tienen en común la cualidad de poder visualizar, manipular y aplicar efectos a la onda sonora y de manejar los formatos más actuales de audio. Entre los más populares en el sector medio y profesional están *Sound Forge* o *Cool Edito Pro*. En nuestros días, es muy frecuente que cada tarjeta de sonido del mercado proporcione un programa de edición de audio. Incluso en Internet podemos encontrar este tipo de herramientas a precios realmente asequibles y en ocasiones gratis (opción que aprovecharemos para elegir las herramientas para hacer música y los efectos de sonido de nuestro juego).

¿QUÉ PROGRAMAS UTILIZAREMOS?

Como ya hemos diseñado, *Zone of Fighters* llevará música, efectos de sonido y voces. Para la música principal vamos a utilizar un secuenciador que nos permita mezclar pistas MIDI y pistas de audio al mismo tiempo. Podríamos elegir *Cubase* pero su precio es algo elevado, así que una buena opción son los programas shareware, muy asequibles y con bastante calidad. Encontramos uno que se ajusta perfectamente a nuestras necesidades, el *Anvil Studio*. Esta aplicación nos permitirá reproducir, grabar, componer y editar pistas usando ficheros de música en formato MIDI y pistas de audio en formato .WAV. También nos permitirá componer pistas rítmicas con nuestras propias muestras de sonido.

Otra forma de hacer la música para utilizarla en nuestro juego es por medio de secuencias de muestras. La mejor forma para realizarlo es usar un secuenciador tracker que nos permita utilizar el formato .X3M para almacenar nuestra música, y una buena opción es *FastTracker*.

Por último, necesitamos un editor de muestras para realizar los efectos especiales de sonido. Hemos elegido un conocido programa shareware muy fácil de usar, potente y con la posibilidad de manejar ficheros .MP3; nos referimos al *Goldwave*. Esta herramienta incorpora multitud de efectos para nuestras muestras, cambios de formatos y un sistema de generación de efectos personalizado por medio de fórmulas matemáticas.

En el próximo número...

... conoceremos las herramientas más utilizadas para componer el sonido de un videojuego y cuál utilizaremos nosotros para los efectos especiales y música de *Zone of Fighters*.

Conceptos básicos de Blitz 3D

Al igual que ocurre con el habla o con cualquier tipo de escritura, en la informática existen unas reglas determinadas que se deben seguir para su comprensión por otros individuos.

Un ordenador sólo entiende de ceros o unos, pero este sistema es muy difícil de comprender por el ser humano. Por este motivo, se han inventado multitud de lenguajes diferentes que interpretan estos ceros y unos para poder comunicarse con el ordenador de una forma más coherente y sencilla. Esta experiencia comenzó con el *lenguaje ensamblador*. A lo largo de la historia de la informática, estos lenguajes han aumentado en diversidad, utilización y potencia. Sin embargo, sólo algunos han obtenido la fama necesaria para prevalecer, ya sea por su facilidad de aprendizaje o por su uso a escala mundial. También es importante tener en cuenta a qué sector de la informática están destinados estos lenguajes dependiendo de su arquitectura. Así, por ejemplo, en Inteligencia Artificial dominaron el *Prolog* y el *Lisp;* en el aprendizaje de la programación tuvo las riendas el

Pascal, por su estructuración, y el lenguaje *C*, que actualmente es el más utilizado en todo el mundo, incluso para construir otros lenguajes; y, por último, destacar quizás el lenguaje de programación más popular de todos, debido a su sencillez de uso, el *Basic*.

Al margen de todos estos lenguajes principales o básicos nacieron otros muchos, consecuencia de las necesidades del mercado y del avance informático. Generalmente, son variaciones de sus abuelos o simplemente cambios en su sintaxis o actualizaciones de su vocabulario. A causa de este avance, casi todos buscaron sistemas visuales más cómodos y potentes para trabajar. Por ejemplo, el *C* evolucionó a un sistema de programación orientado a objetos. Por su parte, el primer *Basic* de PC se transformó en un sistema visual orientado a eventos para Windows.

El lenguaje *Basic* siempre ha sido el preferido para la gente deseosa de iniciarse en el mundo de la programación, ya que su sintaxis se acerca mucho al lenguaje coloquial y además permite un uso general. Con el auge de la imagen, el sonido y los videojuegos –en el ordenador personal–, aparecen en escena multitud de nuevos lenguajes de programación dirigidos hacia la gente que quiere desarrollar aplicaciones multimedia y lúdicas *amateur* y semiprofesionales. La característica principal de estos nuevos lenguajes es su sencillez de aprendizaje y su relativa potencia. Ya en tiempos del Atari se dieron a conocer el *Stos,* el *Amos profesional* para ordenadores Amiga o el *Div Game Studio* para PC. También aparecieron aplicaciones programables para desarrollar juegos como *Klik and Play, 3D Construction Set* o el moderno *3D Game*

NOTA

Un programa básicamente está formado por una lista de instrucciones o comandos que el ordenador lee e interpreta secuencialmente dando lugar a imágenes o sonidos

NOTA

El compilador del *Blitz3D* es el encargado de transformar todo el listado de instrucciones escritas con el editor en código máquina, es decir, en el lenguaje que entiende el ordenador: ceros y unos

Studio. Actualmente, el número de nuevos lenguajes orientados a programar juegos y basados en la sintaxis del Basic es extensa. Aun así, *Blitz3D* destaca por encima de todos.

Blitz3D, sucesor de Blitz Basic, como su propio nombre indica, está basado en el *Basic*, aunque también lleva influencia de otros lenguajes como *C* o el antiguo *Amos*.

Para escribir un programa en *Blitz3D* (B3D) hay que tener en cuenta ciertas reglas gramaticales para que el intérprete del lenguaje pueda leerlo y pasarlo al compilador sin problemas. Antes de seguir, hay que aclarar que llamaremos "listado" al conjunto de instrucciones que forman el programa.

COMENTARIOS Y PALABRAS RESERVADAS

Generalmente, resulta útil el uso de tus propios comentarios en el listado de tu programa. Estos comentarios sólo tendrán valor explicativo en el conjunto de instrucciones y serán ignorados por el compilador. Para insertar comentarios en *B3D* se utiliza el símbolo ";". Se puede insertar cualquier tipo de símbolo como comentarios al principio de una línea de código o al final de sentencias, por ejemplo:

```
Function Visualizar_terreno()
....
End Function
Esto también es válido:
Function Visualizar_terreno();
Aquí empieza la función de
visualizar el terreno
....
End Function
```

Las palabras reservadas son aquellas que son utilizadas por el propio lenguaje y no pueden servir para definir algún tipo de dato por el programador. Por ejemplo, no podemos definir print=3 porque print es un comando del lenguaje para imprimir en pantalla caracteres. De todas formas, el editor indica cuándo escribes una palabra o signo reservado, ya que estos adquieren el color azul.

DATOS, IDENTIFICADORES Y TIPOS DE DATOS

Los datos son las posiciones de la memoria del ordenador que contienen los valores que utiliza el programa. Para que el programa pueda utilizar estos valores almacenados en memoria, es necesario que a los datos se les asigne un nombre; ese nombre se denomina *identificador*.

Por medio de estos identificadores podemos dar nombre a variables, constantes, funciones, etc.

Hay ciertas reglas para definir un identificador. Siempre debe empezar por un carácter alfabético, aunque después esté precedido por números o símbolos; por ejemplo los siguientes identificadores son válidos:

```
Vidas
Vida_jugador
E_1
Explosion_1
M_
multi_jugador_1
```

Resulta indiferente utilizar mayúsculas y minúsculas, así *Vidas, vidas* o *VIDAS* es lo mismo. También *B3D* puede identificar nombres iguales para tipos de datos distintos; por ejemplo, si utilizas el nombre *Vida* para definir una variable y el mismo nombre para definir una función en el mismo programa, *B3D* sabrá en cualquier momento a qué te estas refiriendo.

Hasta ahora hemos mencionado los términos "variables", "constantes", etc. Pero ¿qué son?

Sencillamente, son tipos de datos que se comportan y almacenan la información de distinta manera. Básicamente, en *B3D* hay cuatro *tipos de datos* diferentes según su funcionamiento: las *constantes*, las *variables*, los *arrays* o *matrices* y las *estructuras*. Y tres tipos según la información que almacenan: valores numéricos *enteros*, numéricos *flotantes* y valores alfanuméricos o *texto*.

Los valores numéricos *enteros* o *integer (Int)* no tienen parte fraccionaria o decimales; por ejemplo, son números enteros el 3, el 23 o el -42. *B3D* soporta rangos enteros desde -2147483648 hasta +2147483647.

Los valores de *punto flotante* o *float* (*Float*) incluyen parte fraccionaria y nos ayudan a definir operaciones matemáticas con más precisión. El decimal se pone como punto; por ejemplo, 20.3, -2.5, .45 (0.45), son valores

numéricos de coma flotante. Un detalle a tener en cuenta es que este tipo de datos es más lento de procesar que el valor numérico entero.

Los valores *alfanuméricos* o *strings (Str)* son realmente cadenas de texto. Un dato alfanumérico puede contener cualquier tipo de carácter incluyendo los números (estos números serán tratados como caracteres y no como valores matemáticos) y van entre comillas. Varios ejemplos de valores alfanuméricos podrían ser: "Jugador 1", "FIN" o "Sector 23".

VARIABLES Y CONSTANTES

Las *variables* son los tipos de datos más simples. Son nombres o identificadores que el programador asigna a una porción de memoria donde almacenará valores numéricos o caracteres.

B3D sólo puede identificar automáticamente el tipo dato numérico entero, no obstante, necesita ser avisado en los tipos numéricos flotantes y alfanuméricos; para ello, es necesario añadir un símbolo a continuación del identificador: "$" para texto y "#" para coma flotante.

Por ejemplo, en la declaración *Vida=3*, *B3D* entiende que "vida" contiene un valor entero. Aunque se podría poner también como *Vida%=3*. Para definir el mensaje "Sector 23 abierto" sería: Mensaje$="Sector 23 abierto". Y un ejemplo de declaración numérica en coma flotante sería: X#=120.45

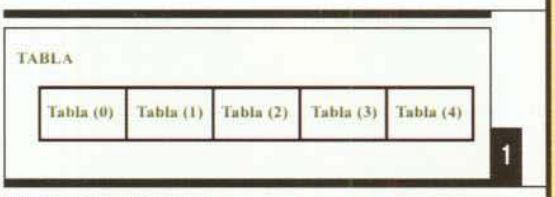
Estos símbolos sólo son necesarios la primera vez que se define el tipo de variable. No se puede elegir el mismo identificador para dos variables de distinto tipo, por ejemplo, es incorrecto elegir el identificador jugador$ y jugador%, ya que el primero sería una variable alfanumérica y el segundo una variable de tipo entero.

Para asignar un valor a una variable se utiliza el símbolo "="; por ejemplo, para asignar el valor *12* a la variable *puntos* se escribiría puntos=12 o puntos%=12 (Hemos reservado un sitio en la memoria con el nombre "puntos" y hemos guardado allí el valor numérico entero "12").

La denominación de variables viene determinada porque los valores que contienen pueden cambiar durante la ejecución del programa. Si definimos en el principio del listado la variable *vida=3*, estamos diciendo que el jugador tendrá al comenzar la partida tres vidas. Pero, en el caso de que pierda 1 durante el juego, esta variable debe cambiar su valor a 2; para realizar la resta basta con poner *vida=vida-1*. Esta misma operación nos servirá para cuando el jugador pierda otra vida más.

EL ALCANCE DE LAS VARIABLES

Al hablar de "alcance" nos referimos a hasta dónde puede llegar el valor contenido en una variable dentro del programa. *B3D* trabaja con dos tipos de variables según su alcance: globales y locales.



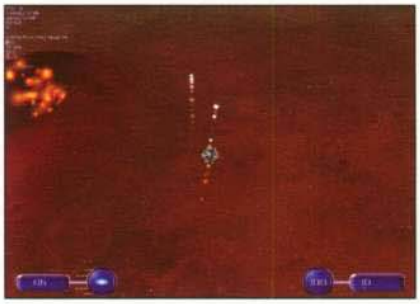
Array de una dimensión.

Las variables globales tienen un alcance global y pueden ser usadas en cualquier parte del programa, es decir, que a esta posición de memoria definida como "global" se puede acceder desde cualquier sitio del programa. Sin embargo, las variables locales tienen su campo de acción en la función que las declara.

Para declarar variables de tipo global se utiliza la palabra *Global* antes del identificador, por ejemplo, *Global vidas=5*. Y para las de tipo local se utiliza *Local*, por ejemplo, *Local x=100*.

Si al declarar la variable no defines de qué tipo es, *B3D* entenderá que es una variable local.

Las constantes son en realidad variables pero con valores fijos que no serán cambiados durante la ejecución del programa. Adquieren utilidad en aspectos del programa que nunca cambiarían como el valor del seno de 90 o una resolución de pantalla. Para definir una constante se utiliza la palabra "Const"; por ejemplo, la variable constante más típica en un programa es PI, y se definiría al principio del programa como: Const PI= 3.141592



Varias constantes pueden ir separadas por coma en la misma declaración, por ejemplo:

```
Const largo_pantalla=800,
ancho_pantalla=600
```

En *B3D* existen dos constantes predefinidas y son: el valor de PI que es PI y los valores 1 y 0 como *True* y *False* respectivamente.

ARRAYS

Un array o matriz es en realidad un identificador que se relaciona con muchas posiciones de memoria. Estas matrices pueden ser de una o dos dimensiones. (ver figura 1)

Para definir un array se utiliza la palabra reservada Dim seguida del identificador y entre paréntesis la cantidad de celdas de memoria que se le asignará a ese identificador. Por ejemplo, vamos a definir una matriz unidimensional llamada "Enemigos" que contenga 5 celdas de memoria: Dim Tabla(4)

Os preguntaréis por qué 4 y no 5. Esto es debido a que la casilla primera empieza en el 0 y no en el 1, así pues tenemos 5 valores desde 0 hasta 4.

Podemos asignar valores a cada posición de memoria de la tabla en el momento de la declaración: `Dim Tabla(4)=12,3,4,60,100` esto significa que en la posición 0 de Tabla hay un 12, en la 1 de Tabla un 3 y así sucesivamente.

Para pasar un valor cualquiera de la Tabla a otra variable, por ejemplo el valor 60 a "fuerza", escribiríamos: fuerza=Tabla(3), porque la posición 3 (en realidad es la cuarta) contiene el valor 60.

Los arrays bidimensionales son como una matriz dentro de otra matriz. Por ejemplo, definimos la siguiente matriz bidimensional Dim posición_planta(4,2). Estamos definiendo 15 posiciones de memoria para la variable "posición_planta" (0..4 x 0..2 = 5 x 3), pero con un cierto orden. En realidad tenemos una cuadrícula de 5 filas por 3 columnas. En la posición 0 almacenamos 3 valores, en la posición 1 otros 3 y así sucesivamente. Para acceder a los valores o posición, en este caso podrían ser las coordenadas X, Y y Z, de la planta número 3 se escribiría (ver fig. 2):

```
X=posicion_planta(2,0)
Y=posicion_planta(2,1)
Z=posicion_planta(2,2)
```

En el siguiente número estudiaremos con más detenimiento cómo se usan las matrices en *B3D*

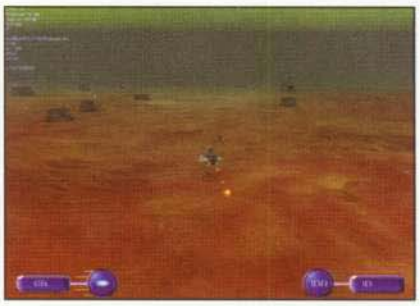
POSICIÓN_PLANTA	X	Y	Z
posición_planta 0	(0,0)	(0,1)	(0,2)
posición_planta 1	(1,0)	(1,1)	(1,2)
posición_planta 2	(2,0)	(2,1)	(2,2)
posición_planta 3	(3,0)	(3,1)	(3,2)
posición_planta 4	(4,0)	(4,1)	(4,2)
posición_planta 5	(5,0)	(5,1)	(5,2)

2

Array de dos dimensiones.

DECLARACIÓN DE FUNCIONES

Las funciones son bloques de instrucciones que realizan una tarea determinada; por ejemplo, podemos tener una función para calcular e imprimir la puntuación del jugador en nuestro programa. De esta forma no tendremos que escribir los mismos comandos cada vez que queramos realizar estas operaciones. Las funciones pueden admitir y devolver valores, utilizándose en muchas ocasiones como procesos para calcular operaciones.



Para definir y abrir una función se utiliza la palabra reservada *Function* más el identificador y los parámetros y para cerrarla las palabras *End Function*. La estructura básica es la siguiente:

```
Function identificador
(parámetros)
   instrucciones
End Function
```

Los parámetros pueden ser una o varias variables separadas por comas que pasan a la función cuando es llamada y tiene que ser de tipo local. Para salir de una función antes de llegar a *End Function* podemos utilizar el comando *Return*, que si lleva detrás una expresión o una variable retornará también su valor.

Las funciones son una cualidad muy interesante del *B3D* y tremendamente útil, que seguramente inundará vuestros listados. No lo dejaremos aquí, ampliaremos el estudio de su uso en la siguiente entrega de esta obra.

En el próximo número...

... continuaremos aprendiendo a utilizar las instrucciones básicas de *Blitz3D*. Estudiaremos, entre otras cosas, qué son las sentencias, el verdadero motor de un programa.

Editor de niveles Q3Radiant (II)

En la primera parte de este tutorial aprendimos a movernos por el *Q3Radiant* y empezamos a crear una sala. Siguiendo con nuestro interior, vamos a añadir cuatro elementos cilíndricos que nos servirán como columnas en cada esquina de la habitación.

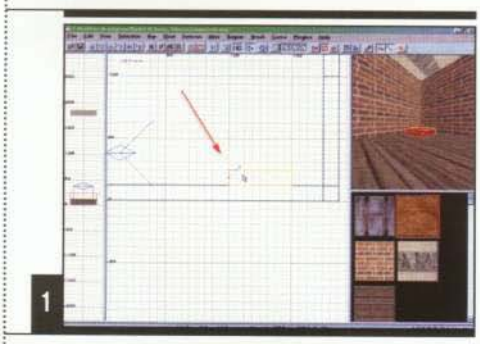
No podemos dibujar directamente en la parrilla primitivas como esferas, cilindros o *brushes* de distinto número de caras. Debemos dibujar primero un cubo, el cual contendrá cualquiera de los elementos anteriores. Por lo tanto, para nuestra columna haremos un cubo cerca de una de las esquinas de la sala, de 62 unidades de lado por 16 de altura para la base y luego crearemos el cilindro eligiendo la opción "Cylindre" en el menú "Curve". Recordad que hay que desplazar la base de la columna en la ventana de alturas y colocarla sobre el suelo.

TRUCO

Para alejar o acercar la cuadrícula se pueden utilizar las teclas *Insert* y *Supr*.

TRUCO

Podemos subir y bajar el objeto seleccionado con las teclas + y - del teclado numérico.



1

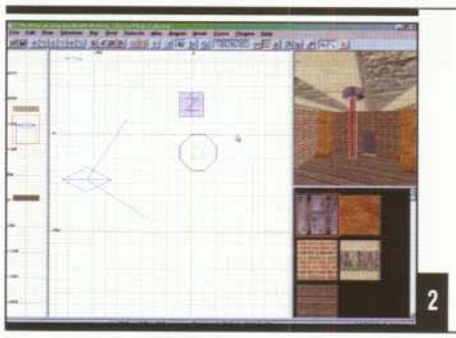
Con el cursor del ratón colocamos los dos puntos de corte.

Nuestra base no será cilíndrica sino que tendrá un pequeño chaflán en su parte superior para suavizar el filo. Este corte lo haremos con la función "clipper". En primer lugar con *CTRL + TAB* elegimos la vista Frontal XZ, pulsamos en el botón "clipper" y con el cursor del ratón colocamos los dos puntos de corte como se muestra en la figura 1.

La parte del objeto que quedará después del corte se vuelve de color amarillo y el trozo que desaparecerá de rojo. Para culminar el corte pulsamos *Enter*.

Podemos ver en la ventana de cámara cómo el chaflán sólo está en uno de los lados. Para el filo que nos queda, cambiamos a la vista lateral (side) YZ. Colocamos los puntos de corte de la misma forma y pulsamos *Enter*. Antes de cortar definitivamente, se puede ver en la ventana de la cámara cómo quedará la figura; eso nos ayudará a no cometer errores. Si los puntos no los hemos colocado correctamente podemos anular la operación pulsando *Escape*. Cambiamos de nuevo a la vista superior y ya podemos mover la base de la columna a la esquina y le asignamos la textura de la piedra.

Seguimos con el cuerpo de la columna, que sí es cilíndrica. Sobre la base dibujamos otro cubo que llegue hasta el techo. A continuación, vamos a darle forma cilíndrica. Podemos crear varios tipos de cilindros dependiendo de la cantidad de polígonos. La más aceptable es la opción "Cylindre" del menú "Curve".



2

Podemos realizar el cambio de tamaño usando el ratón.

Bueno, ya tenemos nuestra columna. Para las otras tres sólo tenemos que duplicar la primera.

Seleccionamos desde la vista de cámara la base y el cuerpo de la columna. Para duplicarla podemos hacerlo pulsando *Espacio* o con *CTRL + C* (Copy brush) y *CTRL + V* (Paste Brush)

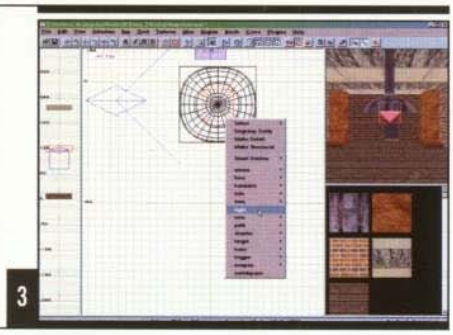
Una vez duplicada, la movemos hacia la otra esquina.

NOTA

En la función "clipper", el trozo del objeto que corte dependerá de dónde se coloque el primer punto de corte. Si lo situamos más arriba que el segundo punto, cortará a partir de su derecha y si está colocado debajo cortará a su izquierda.

TRUCO

Podemos subir o bajar la cámara con las teclas "D" y "C".



Sólo veremos la luz cuando se compile del mapa.

Observamos que, para que encaje, tenemos que girarla 180 grados sobre su eje Z. Para realizar operaciones de rotación o inversión de ejes se utilizan los botones . Para nuestra columna elegiremos el botón . Vemos que cada vez que lo pulsamos gira el objeto seleccionado 45 grados sobre sí mismo (eje Z en vista superior).

Para el resto de las columnas realizaremos la misma operación.

FABRICANDO UNA LÁMPARA CON SU LUZ

Una vez que tenemos cada columna en su sitio vamos a colocar en el techo una simple lámpara con la luz encendida. Para el soporte de la lámpara colocaremos un *brush* de 8 lados.

Dibujamos un cubo pequeño y lo desplazamos hasta el techo; luego seleccionamos la opción "8 sided" en el menú "Brush". Para escalar el objeto libremente se utilizan los botones . Cuando pulsamos el botón de escalado , el objeto se torna de color verde. Desplazando el ratón haciendo *clic*, realizamos el cambio de tamaño (con *Escape* aceptamos la opción).

Para el tubo de la lámpara podemos copiar la base, alargarla y estrecharla. Para duplicar la base no utilizaremos *Espacio*, porque este procedimiento realiza un pequeño desplazamiento. Utilizaremos *CTRL+C* y *CTRL+V*. Una vez duplicada la base, movemos la copia hacia abajo con la tecla del *menos*, la alargamos hacia abajo y la estrechamos con la opción de escalado.

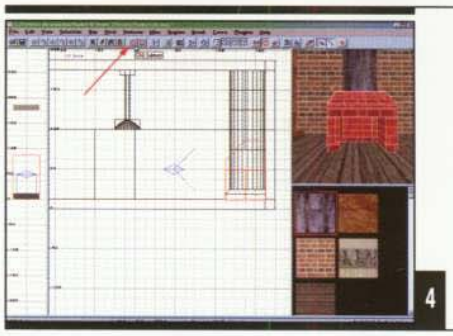
Para fabricar la cabeza de la lámpara vamos a utilizar un cono. Dibujamos el cubo y seleccionamos "Cone" en el menú "Curve" o bien elegimos la primitiva "Cone" en el menú "Brush" y así podemos darle la resolución que queramos.

Una vez fabricada la lámpara, vamos a colocar la luz. Para ello, debemos convertir un pequeño cubo en una luz. Dibujamos el cubo en el centro de la lámpara; con el objeto seleccionado pulsamos el botón derecho del ratón y emergerá un menú. Elegimos la opción "Light" y luego pulsamos *Escape*. Vemos que el cubo se ha transformado en un rombo de color verde. En el editor no podemos apreciar la acción de esa luz. Sólo la veremos cuando se compile el mapa y sea renderizado por el motor del *Quake III*.

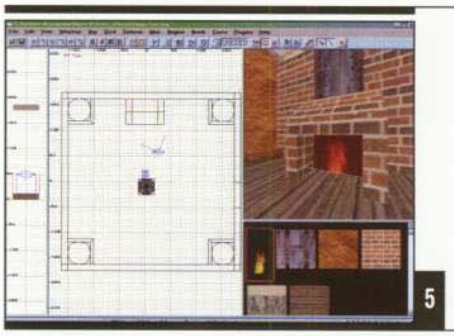
Vamos a terminar nuestra sala colocando una chimenea con fuego. Dibujamos un cubo en la pared y le realizamos un gran chaflán en su parte superior frontal. A continuación, le colocamos la salida de humo y posteriormente con la función "CSG Subtract" le practicamos el hueco donde irá el fuego. El hueco se hará dibujando dentro de la chimenea un cubo con el tamaño del hueco; pulsamos en "CSG Subtract" y borramos con la tecla *BackSpace*. Para el fuego podemos utilizar un "shaders" como efecto de llama.

Debemos sumar una lista de shaders llamada "sfx" a nuestras texturas eligiéndolo en "Load from List" de "Load" en el menú "Textures".

La brillantez de tus niveles dependerá del modelado, diseño y calidad de las texturas que emplees. Este pequeño tutorial ha dejado al lado estas cualidades artísticas y se ha centrado en mostrar los pasos técnicos para comenzar a conocer este estupendo editor BSP.



Dibujamos una chimenea con su respectivo hueco para el fuego.



En el menú "Textures" podemos elegir la opción shaders.

TRUCO

Para desplazar la cámara libremente, nos situamos en la ventana de ésta y movemos el ratón con el botón derecho pulsado.

TRUCO

Si tenemos problemas en ajustar los objetos, siempre podemos reducir la resolución de la rejilla con las teclas del 1 al 7 del teclado no numérico.

En el próximo número...

... arrancaremos con una serie dedicada al sonido. Y la mejor forma de hacerlo es con el secuenciador más popular del mercado: *CUBASE VST*.

Los primeros ordenadores domésticos

Paralelamente al desarrollo de las consolas para juegos, en los 80 aparecieron multitud de microordenadores que proporcionaban una nueva forma de entretenimiento.

Estos sistemas dieron paso al desarrollo de nuevos géneros de videojuegos, con gráficos, sonidos y desarrollos más variados. A excepción de Commodore, prácticamente todos los ordenadores domésticos se basaban en los microprocesadores *Z80* y *Z80-A* de Zilog, de 8 bits de potencia. Destacaron los modelos ZX-Spectrum, MSX y Amstrad. Al principio, estos equipos mostraban sus posibilidades a través de juegos que se regalaban al comprarlos, generalmente versiones de recreativas de la época. A pesar de que el único avance que manifestaban estos ordenadores en muchos años era la ampliación de memoria, los juegos sí mejoraban mes a mes. El sector lúdico empezó a crecer de nuevo gracias al surgimiento de numerosas desarrolladoras y distribuidoras de juegos. Esto ocurrió gracias a la arquitectura abierta de estos sistemas, que permitía a cualquiera que tuviera conocimientos de *BASIC* o código máquina programar su propio juego y conseguirle una distribución. Muchas de las compañías de juegos que hoy conocemos nacieron en esa época. Otros sistemas como los PC basados en procesadores Intel tenían todavía un precio demasiado alto y su utilización para jugar no estaba muy extendida.

Los 8 bits realmente dominaron en el mundo de la informática personal desde los años 80 hasta la década de los 90.

MSX

Este tipo de microordenador se usó hasta casi 1995. MSX nace de la iniciativa de varios fabricantes que buscaban una norma común que permitiese fabricar ordenadores compatibles entre sí. Eran equipos basados en los procesadores Z80A a 3,5 Mhz. Tenían la particularidad de poder ser ampliados mediante cartuchos. Poseían de 16 hasta 64 Kb de memoria para los MSX-1 y hasta 256 Kb para los MSX-2, además de un teclado semiprofesional. Llegaron a ser muy populares, debido también a la calidad multimedia: incorporaban sonido de 3 canales y 8 octavas y gráficos de hasta 256 x 192 con 16 colores a la vez en pantalla en los modelos MSX-1 y 512 x 512 píxels y 256 colores en los modelos MSX-2.

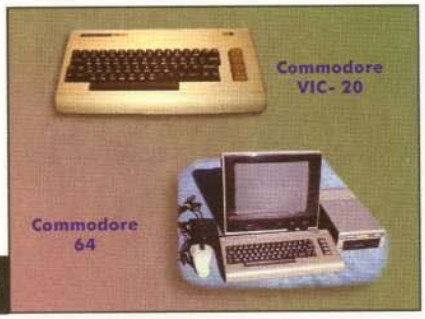
No fueron muy populares en Europa y EEUU debido a un escaso marketing. Aun así, los MSX hicieron furor en sitios tan dispares como Japón o Brasil. Aunque la calidad de sus juegos era muy superior a la de sus competidores de 8 bits, no tuvieron mucho éxito comercial ya que los mejores los incluían en cartuchos, encareciendo con ello el producto.

La posibilidad de asignar hasta 16 colores diferentes a un solo píxel de pantalla, y los sonidos en varios canales, posibilitaban unos juegos realmente atractivos. Desarrolladoras de recreativas como Konami fueron las que más títulos realizaron para

1 *Los primeros ordenadores de 8 bits y algunos ejemplos de cómo eran sus juegos en el comienzo.*

2 *Los ordenadores MSX y MSX2 llegaron a ser muy populares en algunos países.*

3 *Los juegos para MSX tenían una calidad extraordinaria y fueron muy abundantes.*



4

El Commodore 64 y VIC-20 triunfaron en Estados Unidos y eran muy avanzados para la época.

5

Algunas pantallas de juegos para Commodore 64, sin duda una máquina adelantada a su época.

6

Los ordenadores de 8 bits de Amstrad tuvieron mucho éxito en Europa.

NOTA

El Commodore 64 hizo furor en EE.UU., mientras que en Europa los usuarios se decantaron más por el ZX-Spectrum.

MSX. Para MSX-1 destacan juegos como *Zaxxon, Ahtletic Land, Nemesis, Knightmare, Blade Lords* o *Akin*. Los sistemas MSX-2 llegaron a competir con los Commodore Amiga con juegos tan fantásticos como *Super Deform Snatcher* (exclusivamente vendido en Japón) o el *Pennant Race* 1 y 2, juegos que necesitaban la friolera de 64 Kb para funcionar. No podemos olvidar el *Metal Gear* 1 y 2, antecesores del actual *Metal Gear Solid* para PlayStation 2.

COMMODORE VIC-20 Y 64

Commodore VIC-20 surgió en el 1980 y revolucionó el mundo de los juegos para ordenador en EEUU. Ya por entonces disponía de un procesador independiente para el manejo del vídeo. Se llegaron a vender un millón de unidades y se realizaron más de 500 juegos en cassette para él. Evolucionó hasta el Commodore 64, sin duda una máquina adelantada a su época (1982). Basado en el microprocesador 6510 de 8 bits, disponía de hasta 64 Kb de memoria y una resolución de 320 x 200 a 16 colores. Pero lo que hizo potente a este ordenador no fue el escaso megahertzio de velocidad de su micro sino la separación de tareas que proporcionaba su arquitectura, ya que poseía un procesador 6567 que controlaba hasta 64.000 píxels (puntos de pantalla) de vídeo y podía generar 64 *sprites* a la vez y un procesador 6581 para controlar el sonido.

Todos los juegos del momento eran versionados para este ordenador y tenían una calidad envidiable para aquellos años. El éxito del C64 se centró en EEUU. En el mercado europeo lo tuvo más difícil, debido al dominio del ZX-Spectrum. Además, las nuevas desarrolladoras se decantaban por el Z80 por su facilidad de programación.

Hasta que no apareció el Amstrad CPC, los reyes del sector eran los Spectrum y C64.

AMSTRAD CPC

Este ordenador nació en 1984, de manos de Alan Sugar, para competir con el ZX-Spectrum. Estaba basado también en el microprocesador Z80, y tenía incorporado un cassette para cargar y grabar los programas; el equipo completo incorporaba un monitor en color de 14 pulgadas. Gozaba de mayor memoria (64 Kb) y de la posibilidad de utilizar un color por cada punto gráfico de pantalla, lo cual permitía desarrollar juegos visualmente más ricos.

Al igual que C-64, los CPC incluían sendos procesadores para controlar el vídeo y el audio por separado.

Casi todos los juegos realizados para Spectrum y MSX-1 fueron versionados sin dificultad para Amstrad, ya que todos tenían en común el mismo procesador.

Este ordenador evolucionó en el CPC-6128 con 128 Kb que sustituyó la unidad de cassette por la de disco. Poco a poco fue desplazando al Spectrum de Sinclair, cuya compañía terminó comprada por Amstrad, quien siguió comercializando los Spectrum 128 +.

NOTA

Actualmente son muy populares entre los "nostálgicos" los emuladores de los primeros ordenadores domésticos. En el CD-ROM adjuntamos algunos.

En el próximo número...

... hablaremos de la introducción del ZX Spectrum, de su época dorada y del avance tecnológico de sus juegos.

Cuestionario Videojuegos 3

Preguntas

1. ¿Cuáles son los tipos de datos según la información que almacenan?

2. Un edificio tiene 10 pisos y 20 habitaciones por cada piso.
A. Define una tabla para saber la posición exacta de cualquier habitación.
B. Introduce en la variable *"habitación"* la habitación 15 del piso 3

3. ¿Qué se describe en un guión técnico?

4. ¿Cómo debe ser el comportamiento de los personajes secundarios de un juego para ganar jugabilidad?

5. ¿En qué consiste modelar en baja poligonización?

6. ¿Qué es el "Browser" del *Paint Shop Pro* y cómo se activa?

7. ¿Qué es un sampler?

8. Enumera al menos tres secuenciadores para ordenador .

9. ¿Qué procedimiento debemos seguir para añadir una geometría que no sea cúbica en el *Q3Radiant*?

10. ¿Cómo podemos añadir una luz en el *Q3Radiant*?

Respuestas al cuestionario 2

▷ **1.** **a.** F5 **b.** CTRL+ F4 **c.** CTRL + H **d.** CTRL + INICIO / CTRL + FIN
e. INICIO / FIN **f.** CTRL + CURSOR DCHO / CTRL. + CURSOR IZQ
g. CTRL + C **h.** CTRL + V **i.** CTRL + F **j.** CTRL + R

▷ **2.** Se pulsa CTRL + R para ir a la ventana de "Reemplazar". En "Buscar" se escribe la palabra que se quiere cambiar y en "Reemplazar por" la palabra nueva. Luego se pulsa en "Reemplazar todo".

▷ **3.** En un sistema 3D se puede ver la acción desde distintos ángulos en tiempo real, mientras que en dos dimensiones sólo dispondremos de una vista.

▷ **4.** El rendimiento en un videojuego consiste en la cantidad de imágenes por segundo que se consigue. Su importancia radica en mantener esa velocidad constante y superior a 24 frames (imágenes) por segundo para evitar saltos en la acción.

▷ **5.** Para el diseño gráfico 2D, el *Photoshop* de Adobe. Para el modelado y animación en tres dimensiones, el *3D Studio Max* de Discreet.

▷ **6.** En primer lugar se realiza un diseño en papel mediante un boceto, a continuación se construye, modelándolo en un programa 3D. Una vez modelado, se crean las texturas en un programa de dibujo o bien se texturiza directamente en una aplicación especial como *Deep Paint 3D*. Por último, queda definir los movimientos que tendrá el modelo en un programa de animación.

▷ **7.** Generalmente se utilizan los formatos .WAV, .MP3, .X3M, y .MIDI para la música y el formato .WAV, de nuevo, para los efectos especiales de sonido y las voces en off.

▷ **8.** El formato .WAV ocupa mucho espacio en el disco y en memoria, sin embargo, almacena mayor calidad de audio y al no estar comprimido su reproducción es inmediata. Por otro lado, el .MP3 ocupa hasta 14 veces menos capacidad y mantiene una calidad muy similar al formato .WAV, pero tiene el gran inconveniente de que al estar comprimido necesita ser descomprimido en tiempo de reproducción, disminuyendo el rendimiento del juego.

▷ **9.** Las texturas pueden estar en formato TGA de 24 bits de color pero sin canal alfa, o a 32 bits con canal alfa. También se pueden utilizar texturas en formato JPG sólo a 24 bits de color.

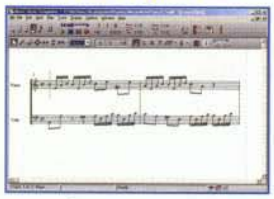
▷ **10.** Esta herramienta se denomina "Surface Inspector" (inspector de superficies) y la activaremos pulsando la tecla "S".

Contenido CD-ROM 3

▶ AUDIO

■ S-Cal 4.1
Calculadora de b.p.m, tiempos de silencio, tiempos de retraso, etc... Muy útil para darle la forma final a nuestra melodía.

■ Music Write 1.0



Herramienta de composición musical muy intuitiva y que incluye multitud de efectos y prestaciones.

■ Dublt 2.0.1
Útil aplicación multimedia que permite integrar sonidos y música en formato .wav a imágenes o clips de vídeo de un modo muy cómodo.

■ AcidPro 3.0
Potente herramienta de creación musical, con una gran capacidad de composición y edición.

■ Acoustic Labs Mixer 2.1
Podemos convertir fácilmente nuestro PC en un editor de audio y en una mesa de mezclas gracias a este programa.

■ Digital Vision DSP 151
Procesador de sonido digital en tiempo real.

■ Cakewalk
Para grabar, reproducir y organizar nuestra música digital con una completísima herramienta.

DISEÑO 2D:

■ Texture Processor 1.3

Podremos crear variadas texturas para nuestras imágenes de un modo sencillo.

■ Object Paint 1.0
Sencilla aplicación que nos será de gran ayuda cuando queramos dibujar formas básicas rápidamente.

■ Fractal Explorer 1.23
Gracias a este programa, podremos crear fractales para realizar los fondos del juego.

■ PictureViewer 1.0.54
Útil herramienta que nos dará una vista rápida de las imágenes que tengamos almacenadas.

■ Neat Image 1.1
Corrector de imágenes y fotografías, para que éstas se vean libres de efectos y brillos indeseados.

■ ThumbsPlus
Programa para localizar, ver y catalogar nuestras imágenes.

▶ DISEÑO 3D:

■ It'sMe 2.0

Divertidísima herramienta con la cual podremos modelar personajes 3D con fotografías de nuestros amigos.

■ CyberMotion 3D-Designer 7.0
Modelado, animación y renderizado de objetos 3D.

■ 3D Reducer 1.1
Optimización de modelos 3D con visualización de éstos en tiempo real.

■ DesignWorkshop Lite 1.8.4
Completo programa que nos proveerá de todo cuanto necesitemos para crear y grabar nuestros propios modelos en 3D.

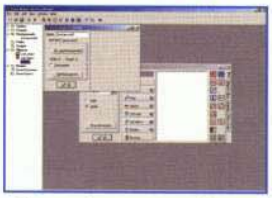
■ ModelMagic 3D
Creación de objetos 3D a base de Open GL.

■ Tutorial texturas Q3 Radiant
Tutorial del Q3 Radiant.

■ Milkshape 3d 1.5.10
Modelador 3D en baja poligonización y fácil de usar.

▶ PROGRAMACIÓN:

■ Game Maker 4.1



Completo paquete destinado a que personas sin conocimientos previos de programación puedan crear un juego desde cero.

■ Digital Mars C/C++ Compiler 8.28
Potente compilador de C y C++ para poder crear programas en entornos Windows.

■ Borland C++ Compiler 5.5
Rápido y optimizado compilador para el lenguaje C++ de 32 Bits.

■ FreePiXCL 4.48
Desarrollo de aplicaciones multimedia de un modo rápido y sencillo.

■ Xtreme Diagram++ MFC Library - MFC 7.0 Compliant 4.20
Permite al desarrollador añadir interfaces gráficas a las aplicaciones Windows.

▶ EMULADORES Y JUEGOS:

Emuladores:

■ WinFrotz 5.3
Emulador de los clásicos juegos de DOS.

■ Mega Drive Emulator 0.99a

Con este emulador podremos convertir nuestro PC en una consola virtual Sega Genesis.

Juegos:

■ Death Zone 1.0
Divertido juego de tipo Arcade con 20 niveles en los que matar malvados aliens.

■ Classic Basic Games 1.0
Conjunto de juegos clásicos de las décadas de los 70 y 80.

■ Zone of fighters
Nueva entrega de nuestro juego Zone of fighters, con muchas mejoras.

▶ VÍDEO:

■ Moviedb 1.3.1.0
Organizador de películas y clips, para que siempre sepamos dónde tenemos cada cosa.

■ MovieXone 4.0.

Una de las herramientas de creación y edición de vídeo más potentes y populares.

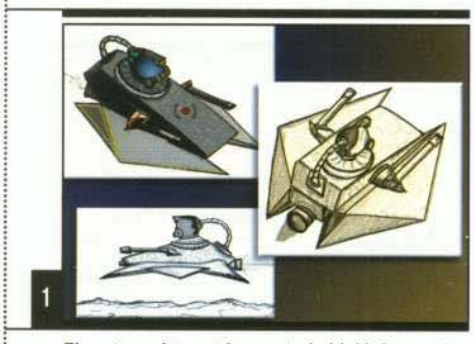
■ Win Effect 1.0
Con este programa podremos crear interesantes efectos de vídeo.

Diseño de los elementos II: los personajes

En el número anterior desarrollamos la idea global de nuestro juego. Realizamos guiones y *storyboards*, dibujamos bocetos y describimos el contenido de la acción. Ya ha llegado el momento de definir cada componente de *Zone of Fighters* con más detalle. Vamos a poner definitivamente en práctica toda la teoría, empezando por nuestro personaje.

EL PROTAGONISTA PRINCIPAL

El protagonista principal diremos que representa el elemento que controla el jugador directamente mediante datos de entrada tomados del teclado, ratón o cualquier otro dispositivo de juego. En nuestro caso, lo que controlamos y vemos en pantalla es un vehículo futurista, una bio-nave de combate; sin embargo, nuestro personaje no es ese vehículo, sino quien lo conduce. De hecho, la historia del juego cuenta que se lucha en naves de combate y no cuerpo a cuerpo. La particularidad del diseño del vehículo hace que se denomine "bio-nave", ya que el guerrero está fusionado físicamente con los controles de la nave. Y para que este concepto tenga un significado visible en el juego, se ha mantenido la cabeza del guerrero (el protagonista principal) en el exterior, recubierta por un casco que está conectado por un tubo "biológico" al casco de la nave (ver fig. 1).



1

El protagonista está conectado biológicamente a la nave.

El porqué de este diseño tiene su explicación en la riqueza de visualización que puede ofrecer.

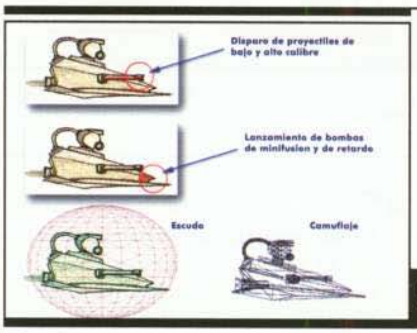
El formato del Blitz3D .B3D, el cual utilizaremos para nuestra "bio-nave", es muy apto para animar modelos orgánicos por la sensación de realidad que ofrece, pero no es nuestro caso. Un vehículo no tiene muchas animaciones con movimientos que puedan dar alguna plasticidad. Sacar la cabeza fuera de la nave e incluir el "bio-tubo" nos permitirá animar el giro de ambos, dando así la sensación de que alguien pilota la nave. Podríamos incluir algunas animaciones más en el modelo como, por ejemplo, el retroceso de los cañones al disparar o mover las alas en los cambios de dirección.

TIPO DE ACCIONES

Básicamente, las acciones que puede realizar una bio-nave están relacionadas con el combate, la defensa y el desplazamiento.

En cuanto al combate y la defensa, puede realizar disparos con sus dos cañones, activar un escudo de protección y camuflarse con el entorno (ver fig. 2).

En cuestión de movimiento, la bio-nave se desplaza en suspensión, flotando a dos o tres metros de la superficie. Por ello, las acciones de aumentar y disminuir velocidad, así como las de cambio de sentido serán suaves y con inercia. Asimismo, para dar un poco más de realismo a la física, aplicaremos por cada cambio de movimiento una ligera inclinación a la nave. A todo esto le sumamos (ver fig. 3):



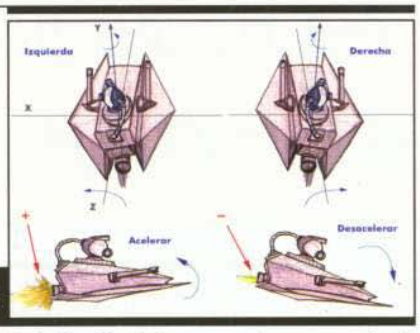
2

Tipos de acciones posibles que puede realizar la bio-nave.

1. Giro de la cabeza a la izquierda o a la derecha de forma aleatoria mientras la nave se desplaza.

NOTA

El formato .MD2 funciona por "Deformación de Malla", consiguiendo una animación suave de las partes de un modelo. Además puede incluir multitud de animaciones distintas en el mismo fichero, sin embargo un modelo en este formato no puede ser detectado por los comandos de colisones del Blitz3D. Por este motivo utilizaremos el formato estándar. B3D, el cual emula perfectamente el funcionamiento de los .MD2 y sí puede ser detectado.



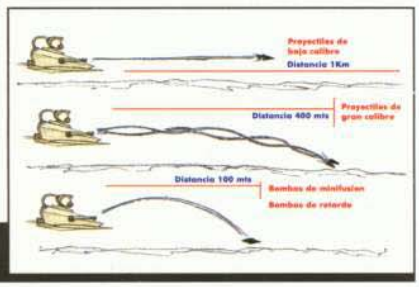
3

La inclinación de la nave aporta un poco más de realismo.

- **2.** En la aceleración aumenta la combustión que sale de la tobera, y al tomar impulso la nave se inclina ligeramente hacia atrás por la inercia.
- **3.** En la desaceleración ocurre lo contrario con la combustión, y la nave, con el impulso, se inclina hacia delante.

A continuación, se muestra una relación más definida de las acciones en el combate de la bio-nave:

- **1.** Está equipada con dos cañones de munición intercambiable; esto quiere decir que puede disparar proyectiles de diferente calibre. Los distintos tipos de munición están repartidos por todo el terreno de combate.
- **2.** Además ofrece la posibilidad de lanzar bombas de retardo, las cuales explotan al cabo de unos segundos después de caer al suelo.
- **3.** Posibilidad de lanzar bombas de minifusión.



4

Comportamiento de los distintos tipos de munición.

- **4.** Está dotada de un sistema de camuflaje que la hace invisible durante cierto tiempo.
- **5.** Capacidad de activar un escudo de protección para repeler ataques durante un tiempo definido.

Los distintos tipos de munición están repartidos por todo el terreno de combate, así como las bombas de retardo. La energía necesaria para activar los sistemas de camuflaje y defensa es limitada, aunque puede ser recargada recogiendo módulos de energía repartidos por el terreno.

(■) TIPO DE ARMAS

Seguidamente, vamos a describir qué clases de armas puede disparar la bio-nave de combate. Realmente, y como ya apuntamos, dispone de dos cañones de munición intercambiable y de un lanzador de bombas de retardo. La cuestión es describir el comportamiento de los distintos tipos de munición (ver figura 4):

- **1.** Proyectiles de bajo calibre. Es la munición estándar. Tienen gran alcance y permiten una frecuencia de disparo elevada, pero su poder destructivo es reducido. No necesita ser recargada, por lo tanto su número es infinito.
- **2.** Proyectiles de gran calibre. Son como pequeños misiles con mayor poder destructivo pero con menor alcance que la munición estándar. Cada recarga tiene un número limitado de 20 unidades, es decir, 10 disparos, ya que cada cañón dispara uno.
- **3.** Bombas de minifusión. Son pequeñas bombas atómicas de 1/10 de megatón cada una. Al ser lanzadas adquieren trayectoria curvilínea ascendente-descendente. Gran poder destructivo pero con poco alcance. Cada módulo de recarga admite sólo dos unidades cada vez.
- **4.** Bombas de retardo. Son granadas con activación temporal, explotan a los pocos segundos de ser soltadas en el aire. Cada módulo puede contener hasta 5 unidades.

Hasta aquí hemos descrito todas las cualidades de la bio-nave de combate que conducirá nuestro protagonista. El siguiente paso será describir todos los enemigos que podemos encontrarnos en el terreno de combate, aparte de los demás guerreros.

Puede ocurrir que otros contrincantes no estén guiados por jugadores humanos, sino conducidos por el ordenador. Para este caso y debido a su extensión, dedicaremos un especial estudio en una posterior entrega.

(■) LOS MALOS DE LA PELÍCULA

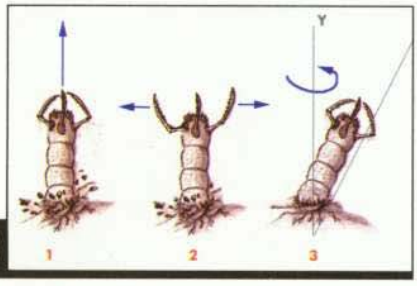
Para dar un poco más de emoción y realismo a la zona de combate, decidimos añadir animales y plantas con comportamientos letales. Procederemos a describir estos seres y su comportamiento.

(■) ANIMALES

Básicamente, sólo incluimos un tipo de animal, lo llamaremos *Slunk*. Son gusanos que habitan bajo la superficie. Son muy peligrosos y aunque son ciegos, tienen un gran olfato y la capacidad de detectar a las bio-naves por la vibración que producen los motores y

5

Los Slunk son gusanos muy peligrosos que habitan bajo tierra.



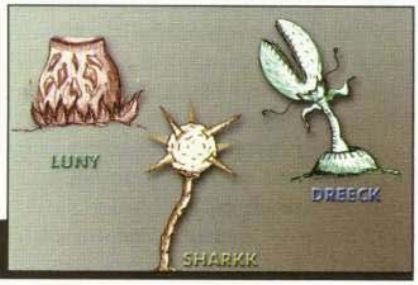
6

Modo de actuar del terrible Slunk.

las explosiones originadas por el combate. Cuando detectan a una presa, salen a la superficie con gran velocidad abriendo sus grandes tenazas. A veces, también salen engañadas por la acción de las bombas, aunque a menudo se dejan ver olisqueando el aire para buscar comida. Han aprendido que con frecuencia la superficie se llena de otro tipo de habitantes (ver fig. 5).

Determinamos pues una inteligencia inferior con un comportamiento bien definido. En momentos aleatorios el *Slunk* saldrá a la superficie, siempre dentro de su hábitat, en tres ocasiones: En una primera, sin motivo aparente y con alguna frecuencia en lugares imprevistos. En una segunda, aparecerá por casualidad junto a alguna que otra explosión. Y en una tercera ocasión, en la trayectoria de la bio-nave.

En todas estas ocurrencias, el *Slunk* hará la misma acción: abrirá sus tenazas y hará un giro en círculo mientras las cierra, para luego introducirse de nuevo en la tierra (ver fig. 6).



7

Tipos de plantas peligrosas.

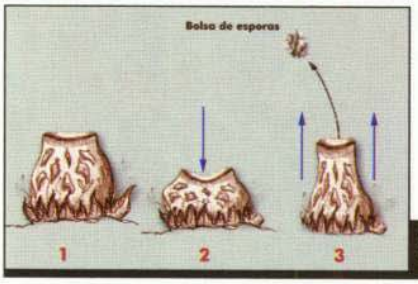
Posee un gran poder destructivo, y un choque en varias ocasiones con el ser puede provocar la total destrucción de la bio-nave. Tampoco es fácil liquidarlo en un enfrentamiento, dado su gran tamaño y la composición blanda de su cuerpo. Aun así, sucumbe bajo el impacto de algunos proyectiles de bajo calibre y con facilidad con un par de gran calibre o cualquiera de las bombas.

Su hábitat preferido es el subsuelo arenoso, debido a la capacidad de movimiento que permite este tipo de material de composición blanda. Sin embargo, y en contadas ocasiones, puede moverse bajo la superficie arenosa del fondo acuático, ya que su respiración cutánea le permite filtrar el oxígeno del agua, así que puede aparecer por cualquier superficie acuática que se encuentre dentro de su territorio.

PLANTAS

En este tipo de seres, incluimos algunas especies que, por su naturaleza, pueden provocar serios daños e incluso la destrucción de nuestra bio-nave. Empezando por los árboles que, dada su robustez, un choque contra ellos dañaría nuestro fuselaje. Sin embargo, hay tres clases de plantas que son particularmente peligrosas: los *Lunys*, los *Sharkk* y las plantas carnívoras *Dreecks* (ver fig. 7).

Los Lunys son plantas que habitan en cualquier superficie arenosa o rocosa. No son carnívoras, pero las hemos incluido como letales por las características especiales de su forma de reproducción. Con frecuencia se contraen para lanzar al aire bolsas gigantes cargadas de esporas a una gran distancia (ver fig. 8). El peligro de esta acción radica en que estas bolsas van cargadas con energía eléctrica con el fin de mantener las esporas unidas. Cualquier intento de separarlas provocará una reacción eléctrica en cadena. Así que un choque con estas bolsas provocará daños en la estructura y en los sistemas de navegación pudiendo incluso hacernos perder el control momentáneo de la nave al cortarse la comunicación biológica entre ésta y el guerrero. Con mucha frecuencia alargan su cuerpo para obtener alimento del aire. Es entonces cuando resultan casi invulnerables, debido a la flexibilidad y resistencia de su cuerpo. Un choque con su piel, también cargada de electricidad, provocará la destrucción completa de la bio-nave.

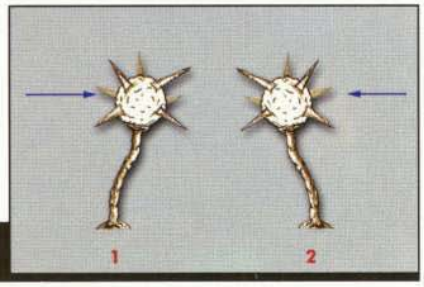


8

Lanzamiento de bolsas de esporas de un Lunys.

Estos seres tienen un comportamiento bastante simple. De forma aleatoria y sin ningún motivo, se contraen para luego estirarse y lanzar la bolsa de esporas. Estas bolsas ascienden a una altura nunca igual y caen a una distancia no superior a 200 metros alrededor de la planta.

Nuestro segundo tipo de vegetal animado son los *Sharkk* o plantas martillo. Son de considerable altura y muy abundantes, tanto que a veces pueden formar pequeños bosques. Su estructura es muy particular. Debido a que su delicado sistema reproductivo está ubicado en el centro de la estructura esférica que está sobre el tallo, posee algunos métodos defensivos para evitar cual-



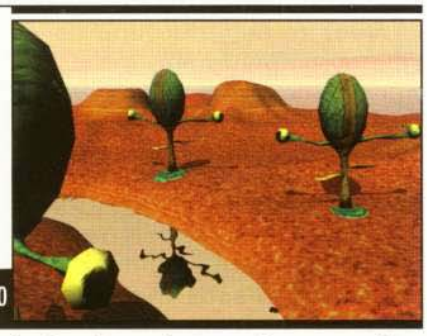
9

Los feroces Sharkk o plantas martillo pueden destruirnos por completo.

quier contacto ajeno con esta estructura. En primer lugar, unos terribles pinchos la rodean provistos de un sistema de detección de vibraciones en el aire. Al menor movimiento del aire a su alrededor, provocará una expansión y contracción del tallo, que origina un golpe en forma de martillo. Un impacto con los pinchos de esta planta provoca daños en la estructura, y varios golpes seguidos pueden causar la destrucción completa de la bio-nave (ver fig. 9).

Son fáciles de destruir, basta con algunos disparos de bajo calibre, pero la expansión de las esporas recluidas en su interior puede provocar un crecimiento masivo en un lugar cercano.

Nuestra última criatura es quizás la planta más peligrosa: los *Dreecks*. Estas plantas son carnívoras, se alimentan de todo lo que pase por su lado. En realidad son un híbrido entre animal y planta. Poseen tallo y hojas, como todo vegetal, pero están provistas de ojos, boca, estómago, y lo peor de todo: cerebro. A pesar de su gran tamaño, tienen una agilidad increíble y siempre están en movimiento. Al ser también plantas, dependen de las raíces para sustentar su parte vegetal, por eso no pueden desplazarse. Sin embargo, su parte animal tiene un apetito terrible y están en continua búsqueda de presas, girando el tronco y moviendo los apéndices oculares (ver fig. 10).



10

Los Dreecks son plantas carnívoras muy peligrosas por su voraz apetito.

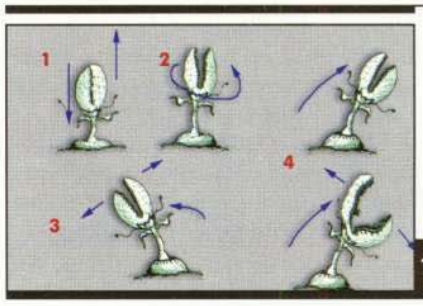
Su posible inteligencia se limita a la posibilidad de describir los elementos que les rodean. Así, pueden seguir la trayectoria de un objeto en movimiento o acechar a cualquier cosa nueva que no esté grabada en su cerebro, la cual consideran como posible alimento.

Sabiendo esto, podemos definir sus movimientos de la siguiente forma:

- **1. Reposo.** Movimientos de arriba a abajo con leves inclinaciones de cabeza.
- **2. Búsqueda de presas.** Mueve el tronco de un lado a otro a la misma vez que mueve sus tentáculos oculares.
- **3. Acecho.** Inclinado y abriendo y cerrando la boca.
- **4. Ataque.** Movimiento brusco del tronco hacia la presa a la vez que realiza un gran mordisco (ver fig. 11).

Un tropiezo con esta criatura letal significa la muerte en el acto. La bio-nave quedará hecha añicos irremediablemente. Destruirla requiere un gran esfuerzo de munición, así que lo más sensato en caso de toparse con ella es esquivarla.

Si bien suelen vivir más o menos en grupos, los individuos están algo distanciados entre sí, y algunas veces se encuentran totalmente solos en lugares dispares fuera de su hábitat normal. Prefieren la arena, aunque no les desagradan las zonas verdes o las laderas de los ríos y lagos.



11

Movimientos de ataque de los Dreecks.

Ya tenemos, de una forma bien definida, el aspecto y todo el comportamiento de nuestra bio-nave y de los demás habitantes de la zona de combate. Sabemos dónde viven los animales y plantas, qué terreno prefieren, qué daños causan y qué resistencia tienen y cómo se comportan.

Todo este diseño nos servirá para poder darles vida luego en programación, además de saber dónde situarlos en el momento de construir el escenario del juego.

(■) OTROS ENEMIGOS

Para añadir aún más emoción y dinamismo al juego, se podría añadir otro tipo de obstáculos enemigos como artefactos voladores que participen en los combates. Estarían esparcidos por todo el terreno y siempre al acecho. Los llamaremos "voladores". Poseen armas de bajo calibre y alcanzar a uno de estos aparatos proporciona puntos extras.

En el próximo número...

... seguiremos diseñando los elementos que componen *Zone of Fighters*, y le llega el turno a todo lo concerniente al entorno del juego: ambientación, decorados, objetos de adorno y comportamiento de los objetos especiales que nos ayudarán a combatir.

Modelado en baja poligonización con Milkshape3D (II)

En el capítulo anterior aprendimos los fundamentos para trabajar con *Milkshape3D*. Continuamos la lección realizando algunos ejercicios para practicar el modelado en baja a partir de primitivas y mediante el sistema de modelado por vértices. La aplicación se encuentra en el CD-ROM.

MODELANDO EL CUERPO DE UN AUTOMÓVIL POR VÉRTICES

El sistema de modelado por vértices es algo lento y tedioso, pero se puede construir cualquier cosa y controlar mejor el número de polígonos. Vamos a tomar contacto modelando -de una manera básica- el casco de un automóvil.

En primer lugar, realizaremos la silueta lateral del casco colocando vértices en la vista lateral derecha como se muestra en la figura 1. Para situar vértices pulsamos la opción "Vertex" del panel de herramientas.

Una vez dibujada la silueta, procederemos a unir cada vértice entre sí para formas las caras.

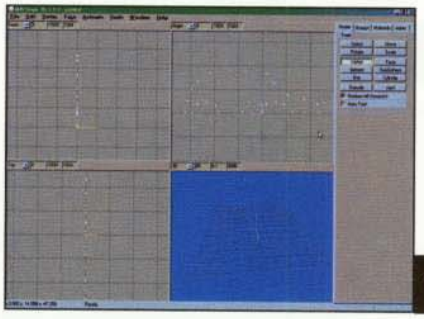
Pulsamos en "Face" en el panel de herramientas, y pulsamos en tres vértices consecutivos según el orden que mostramos en la figura 2 para ir formando los triángulos. En ocasiones puede ocurrir que nos equivoquemos en la secuencia; entonces podemos ver en la vista 3D cómo la cara se dibuja de color oscuro. Esto significa que el orden de las caras está al revés. Una solución es elegir la opción "Reverse Vertex Order" en el menú "Face" o bien deshacer con *CTRL + Z* y volver a construir la cara. Aconsejamos esto último, para ir adquiriendo soltura.

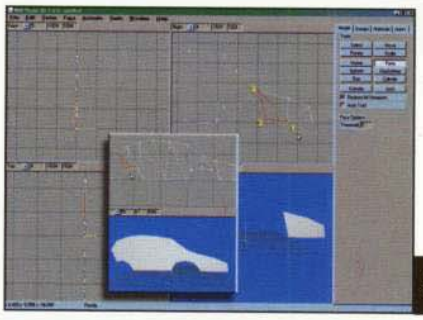
Una vez formada la silueta, observamos que nuestro modelo es plano, debido a que hemos trabajado en una de las vistas. Para dar forma en 3D, vamos a realizar un *extrude* de todas las caras hacia la derecha. Elegimos "Select" (F1) y "Face" en opciones, y con un recuadro seleccionamos todas las caras en la vista lateral. A continuación elegimos la herramienta "Extrude" y nos situamos en la vista frontal para desplazar el ratón hacia la derecha con el botón izquierdo pulsado, como vemos en la figura 3.

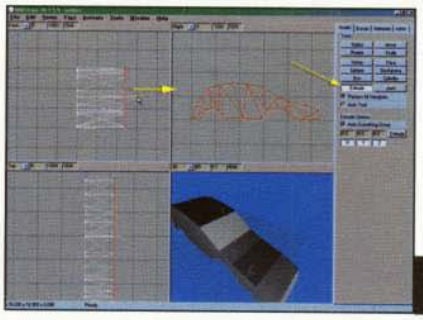
Ya tenemos el cuerpo de medio vehículo. Para la otra mitad sólo tenemos que seleccionar todo, duplicar y realizar un espejo lateral de la copia; es decir, colocar la mitad copia a continuación de la original pero con simetría inversa. Para realizar las simetrías tenemos las opciones de "Mirror" situadas en el menú "Vertex". Es importante colocar el extremo interior de la pieza original a duplicar, alineado con los ejes, para que la operación de simetría se realice con precisión, como en la figura 4. Muy bien. Duplicamos con *CTRL +D* y realizamos una simetría de izquierda a derecha con "Mirror Left < — > Right". Ahora nos queda unir las dos partes. Seleccionamos los vértices centrales con "Ignore BackFaces" deseleccionado y elegimos la opción "Weld Together" en el menú

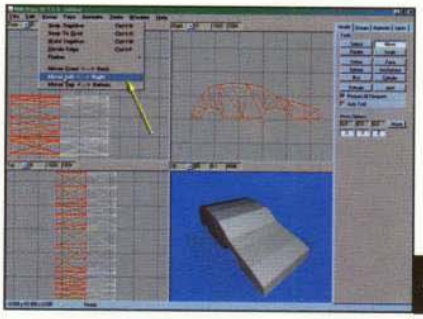
TRUCO

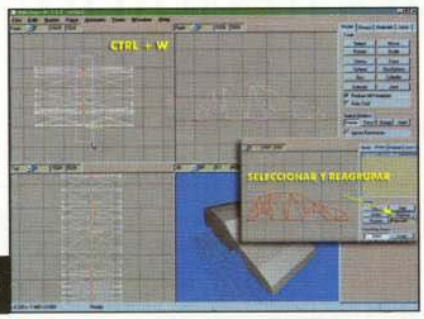
Para cambiar el orden de los vértices podemos elegir la opción "Reverse Vertex Order" en el menú "Face".


1

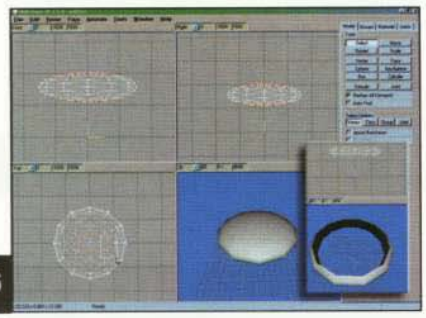

2

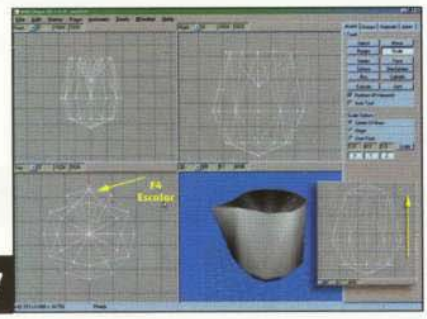

3


4

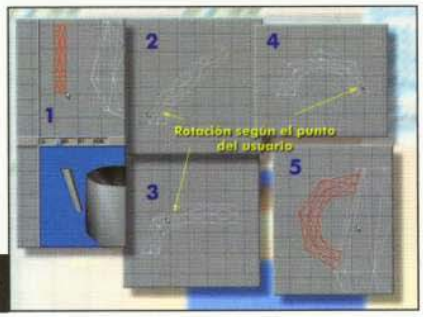


5

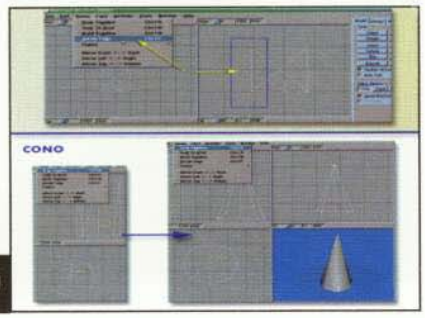

6



7



8



9

"Vertex" o *CTRL + W* para realizar la unión. Pero todavía seguimos teniendo dos objetos diferentes, aunque unidos entre sí. Para convertirlos en un solo modelo debemos reagruparlos. Elegimos la opción "Group" de la herramienta "Select" (F1) y seleccionamos las dos partes. Pulsamos en la pestaña "Group" de la ventana de herramientas y pulsamos sobre "Regroup". Esta operación une las dos partes, reagrupándolas en una, tal y como vemos en la figura 5.

OPERACIONES CON PRIMITIVAS

Para terminar nuestra preparación, vamos a modelar pequeños objetos a partir de primitivas para poder practicar todas las herramientas y funciones de modelado.

En primer lugar, vamos a construir un anillo a partir de una esfera de 12 partes (*stacks*) y 12 divisiones (*slices*). Pulsamos la herramienta "Sphere", y en "Sphere Options" elegimos el tamaño. Una vez dibujada, pasaremos a realizar un escalado en su eje Y para achatarla. Pulsamos en "Scale" (F4) y en la casilla Y de opciones escribimos el valor 0.3 y pulsamos en "Scale". Esta forma es más exacta que escalar la esfera manualmente desde cualquiera de las vistas. Para el agujero del centro, nos situamos en la vista superior (Top) y seleccionamos los vértices centrales poco a poco (utilizar la tecla *SHIFT* para sumar selección) con "Ignore BackFaces" desactivado, como vemos en la figura 6. A continuación borramos (*SUPR*) y ya tenemos el agujero.

Vamos a continuar practicando, modelando una jarra.

TRUCO

Podemos sumar selecciones, manteniendo la tecla SHIFT pulsada mientras seleccionamos.

Partimos de una esfera de 4 stacks y 12 slices (4 x 12). Seleccionamos los vértices superiores y los desplazamos en la vista lateral hacia arriba. Luego, desplazamos hacia abajo el vértice de la unión superior para simular un hueco interior. Vemos en la figura 7 cómo para la boquilla seleccionamos tres vértices de uno de los bordes en la vista lateral y en la vista superior los escalamos (F4) hasta dar la forma. El asa la haremos separadamente y luego la colocaremos en la parte trasera del cuerpo.

Partimos de un cilindro de 4 x 6. Para dar la forma curva debemos seleccionar los vértices de cada parte del cilindro y rotar según el punto del usuario, como se muestra en la figura 8. Una vez curvado, seleccionamos toda el asa y la rotamos y movemos hasta el cuerpo de la jarra.

A veces ocurre que necesitamos tener más vértices de la cuenta en una primitiva, para poder modificarla y obtener formas más complejas. Para solucionar este problema existe una función que divide dos vértices seleccionados, colocando otro en medio, como vemos en la figura 9.

Dibujemos un cubo. Seleccionamos dos de sus vértices y elegimos la opción "Divide Edge" (*CTRL + P*) del menú "Vertex". Vemos cómo aparece un vértice entre los dos seleccionados y se ha formado otro triángulo nuevo.

Para terminar, vamos a crear un cono a partir de un cilindro para aprender a utilizar la función destinada a unir varios vértices en uno.

Dibujamos un cilindro de 1 x 8 y seleccionamos todos los vértices superiores. A continuación elegimos la opción "Snap Together" (*CTRL + P*) en el menú "Vertex".

La mejor manera de dominar *Milkshape3D* es practicando mucho. En el próximo número haremos una pausa para dedicarnos al grafismo 2D de lleno, porque vamos a empezar a trabajar en nuestro juego.

MODELADO 2D

HERRAMIENTAS DE PAINT SHOP PRO 7

En la entrega anterior, tuvimos nuestra primera toma de contacto con *Paint Shop Pro* (incluido en el CD-ROM). A continuación vamos a conocer la paleta de herramientas y todas sus opciones.

En la "paleta de herramientas" se encuentran todas las herramientas necesarias para pintar, dibujar, seleccionar y retocar imágenes. Asociada a cada herramienta, aparecen sus opciones correspondientes en la ventana "Tool Options". Para empezar vamos a crear una imagen nueva (*CTRL + N*) de 400 x 400 píxels a 72 (píxel/pulgada) de resolución y con fondo negro. Vamos a rellenar la nueva imagen con una textura.

Seleccionamos la herramienta de relleno y en la paleta de colores "Colors" elegimos la segunda textura de la lista en "Styles" (ver fig. 10). Y hacemos clic sobre la imagen.

La herramienta nos permitirá acercarnos o alejarnos en la imagen completa o en el trozo que seleccionemos. (Observad los factores de zoom en la ventana de opciones).

La siguiente herramienta nos permitirá recortar la imagen en el tamaño que hayamos seleccionado. Para activar el recorte debemos pulsar en el botón "Crop Image" de la paleta de opciones. Después de recortar podemos volver a la imagen original haciendo un *undo* (*CTRL + Z*).

Tenemos varias opciones de selección geométrica. Desde el típico cuadrado hasta selección en forma de estrella. Además, podemos elegir una selección con bordes más o menos finos (Feather) y disponer de suavizado de bordes (Antialias).

Asimismo, con la herramienta podemos realizar selecciones a mano alzada, punto a punto, o seleccionar bordes entre dos áreas de distinto contraste de color o iluminación (Smart Edge). (ver fig. 11).

Practica con la imagen estas herramientas. Recuerda que para cerrar una selección basta con hacer dos *clics* seguidos. Observa también que, una vez hecha la selección, el cursor se cambia por el de "mover" cuando estás encima del área seleccionada. Esto significa que si desplazas la selección cortarás el fondo y moverás el área seleccionada.

Paint Shop Pro permite hacer copias (*CTRL + C*) de una selección en otra capa (*CTRL + L*), crear una imagen nueva con sólo el área seleccionada (*CTRL + V*), crear una copia de la selección en la misma imagen (*CTRL + E*) o crear una selección transparente (*CTRL + SHIFT + E*).

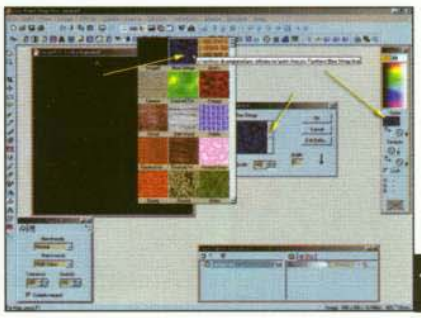
Otra herramienta de selección muy útil es la varita mágica.

Este tipo de selección permite escoger distintas áreas del dibujo basándose en 5 tipos de valores: por cantidad de color rojo, verde o azul (RGB Value); por tono de color (Hue); por luminosidad o cantidad de blanco (Brightness); selección de áreas no transparentes (All Opaque); y por la opacidad de los píxels (Opacity). También permite controlar la similitud entre píxels (Tolerance) y el grosor del borde de selección (Feather).

La opción "Sample Merged" activada indica que se hará la selección en todas las capas, mientras que si está inactiva lo hará sólo en la capa activa.

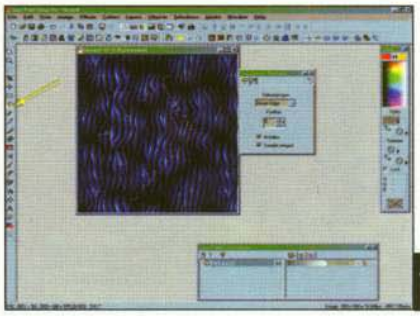
Esta herramienta ("cuentagotas") sirve para seleccionar un color en la imagen. En herramientas como "pincel" o "bote de relleno", podemos activar momentáneamente el cuentagotas manteniendo pulsada la tecla *CTRL*.

Antes de ver las herramientas para pintar, vamos a practi-



10

Podemos seleccionar la segunda textura deseada.



11

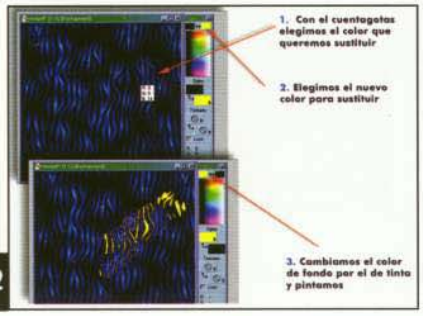
Podemos realizar distintos tipos muy precisos de selecciones.

TRUCO

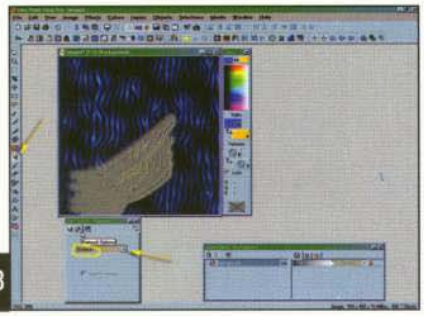
Para trabajar mejor, podemos ocultar la paleta de herramientas pulsando la tecla "P" y seleccionar las herramientas desde la ventana "Tool Options" pulsando en el icono .

NOTA

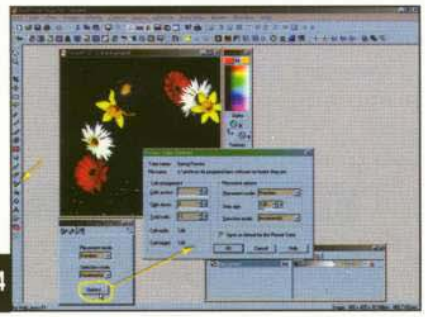
***Paint Shop Pro* posee un historial de "hacer y deshacer" situado en el menú "Edit" llamado "Command History" (CTRL + SHIFT + Z), que nos permite ir hacia atrás o hacia delante en todas las acciones que hayamos realizado.**



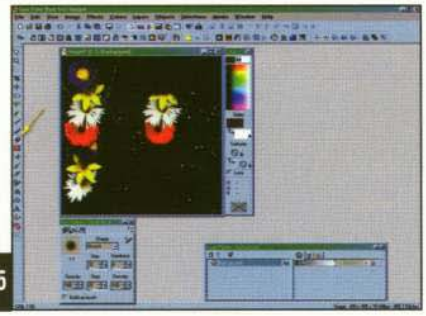
12 *Conmutamos el color de fondo por el de tinta.*



13 *Existen hasta 19 modos distintos de retoque.*



14 *Gracias a las opciones, podemos controlar cómo pintar las imágenes.*



15 *Dibujamos una fila de flores.*

car con dos herramientas muy interesantes: reemplazar color y retocar. La primera nos permite reemplazar un color de la imagen que elijamos por otro color. Pulsamos sobre el icono , con *CTRL* pulsado elegimos con el cuentagotas el color que queremos sustituir. A continuación, en la paleta de colores, colocamos en el color de fondo el que reemplazará al elegido anteriormente. Conmutamos el color de fondo por el de tinta y pintamos (ver fig. 12). Para sustituir de nuevo el color, basta con conmutar de nuevo y pintar.

La siguiente herramienta, "Retouch" , realiza cambios en la imagen según vamos pintando. En el panel de opciones en la pestaña "Retouch Options" podemos elegir hasta 19 modos de retoque. (ver fig. 13).

Vamos a continuar viendo las herramientas de pintar: pincel, clonado, pincel con forma, y spray.

Antes de nada vamos a borrar toda la imagen. Seleccionamos todo con *CTRL + A* y borramos con la tecla *Supr (DEL)*. La pantalla se quedará con el color de fondo seleccionado.

La herramienta nos servirá para dibujar como si estuviéramos pintando con un pincel. En el panel de opciones podemos modificar todos las características de la brocha como tamaño, forma, cantidad de pintura o transparencia.

Con esta herramienta podemos pintar con el dibujo que queramos. Al seleccionarla, nos aparece en la paleta de opciones una bola. Pulsando en la pequeña flechita de la derecha accedemos a la librería de imágenes de *Paint Shop Pro*. En la pestaña de opciones, podemos controlar cómo se pintarán esas imágenes (ver fig. 14).

Para probar la herramienta de clonado , vamos a dibujar primero una fila de flores (ver fig. 15).

Seleccionamos la herramienta de clonado. Marcamos con el botón derecho del ratón sobre lo que queremos clonar. Una vez hecho esto podemos pintar en otro sitio de la imagen, en donde se irá copiando a partir del punto que marcamos al principio.

TRUCO

Para deseleccionar una selección en cualquier momento pulsar CTRL + D). Otra herramienta de selección muy útil es la varita mágica .

TRUCO

Para cambiar el color de fondo por el de tinta en la paleta de colores hay que pulsar en las flechas que están entre ambos .

TRUCO

Para aumentar o reducir el tamaño de visualización de una imagen, basta con pulsar las teclas + o –.

TRUCO

Si queremos tener nuestras propias imágenes para pintar con ellas, tenemos que copiar nuestros ficheros de imágenes en el directorio que tiene Paint Shop Pro para ello.

En el próximo número...

... nos quedan por ver las herramientas de texto, dibujo y polígonos, pero lo dejaremos para el próximo número; vamos a utilizarlas en el diseño del logotipo de *Zone of Fighters*.

El programa Goldwave

En el número anterior vimos las herramientas que utilizaremos para desarrollar el sonido de nuestro juego. En estas páginas, vamos a conocer un estupendo y asequible editor de sonido: *GoldWave* (incluido en el CD-ROM).

Con esta aplicación podemos grabar, editar y convertir sonidos, además de extraer en ficheros de audio las pistas de un CD. Puede tener muchos sonidos abiertos a la vez, con la posibilidad de copiar partes de audio de una ventana a otra. Puede convertir a multitud de formatos de audio incluido MP3.

CÓMO EMPEZAR

Al ejecutar *GoldWave* observamos una ventana flotante de control. Aquí podemos controlar la ejecución y grabación de sonido, su volumen, balance y velocidad de reproducción. Asimismo, disponemos de un gráfico del aspecto del sonido en tiempo real. En el botón de propiedades, podemos configurar las opciones de reproducción, grabación, volumen de dispositivos, gráficos y dispositivos instalados.

Vamos a crear un plantilla de sonido nueva. Para ello, pulsamos en el botón "New". Aparecerá una ventana de diálogo para elegir las características del sonido. Vamos a elegir calidad "CD" y pulsamos "OK". Se abre una ventana de sonido. Si pulsamos de nuevo en "New" se abrirá otra nueva ventana de sonido y así todas las que nuestro sistema permita (ver fig. 1).

Para conocer las herramientas de este programa, vamos a extraer una pista de un CD de música.

Cerramos todas las ventanas anteriores. Introducimos un CD de música y elegimos la opción "CD audio extraction" del menú "Tools". Elegimos nuestro CD ROM y la pista que queremos extraer en la ventana "From". A continuación (para no extraer toda la pista) en "To" seleccionamos el final de la pista para reducir la longitud del fichero (unos 15 segundos). Pulsamos en "Save" y saldrá una ventana que nos pide dónde guardaremos el audio extraído. Escribimos "prueba" y "Guardar". (La extracción comenzará) (ver fig. 2).

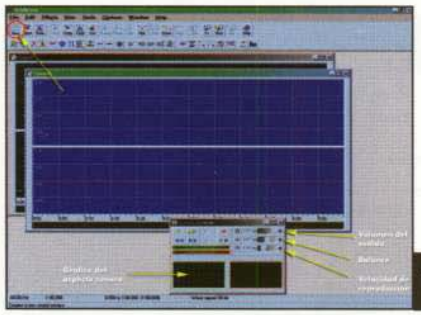
Bien, ya tenemos nuestros 15 segundos de música en la ventana de audio. Pulsad para reproducirla y probad a manejar los controles en la ventana de control. (Ver figura controles).

EDITAR UN SONIDO

Vamos a aprender seguidamente a seleccionar, cortar, copiar, insertar y manejar la ventana de audio.

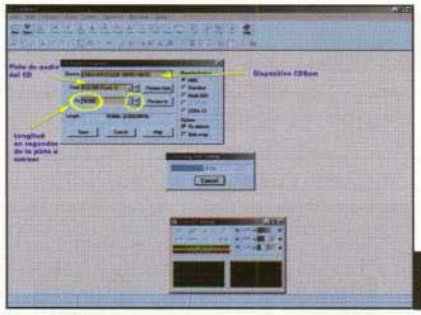
En primer lugar, para ver toda la onda de sonido en la ventana de audio pulsamos en el icono para ajustar la vista.

Para borrar un trozo del sonido, por ejemplo, el silencio del principio de la onda, debemos seleccionar el trozo primero. Moviendo el ratón mientras hacemos *clic* con el botón izquierdo, desplazamos la marca de comienzo de selección y si lo hacemos con el botón derecho moveremos la marca de final de selección. Seleccionemos pues el comienzo de la onda sin sonido. Nos acercamos a la selección para ajustarla correctamente pulsando en el icono. Desplazamos la barra deslizado-



1

Podemos abrir todas las ventanas de sonido que queramos.

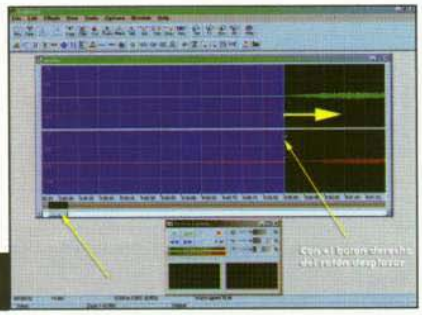


2

Extracción de una pista de un CD de música.

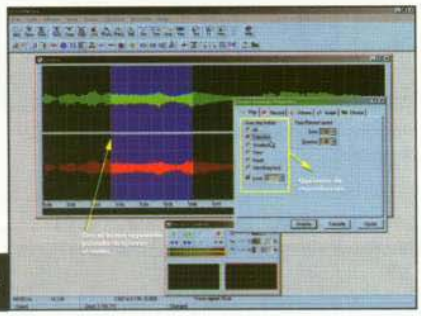
NOTA

Si el sonido es estéreo, la ventana se divide en dos partes. La parte superior corresponde al canal izquierdo y la inferior al derecho. Para conmutar cualquiera de los dos canales hay que hacer clic en el icono .



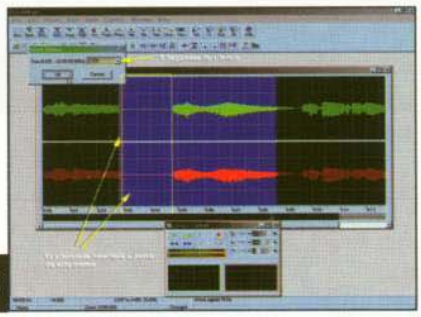
3

Podemos editar el sonido y cortar los silencios a nuestro antojo.



4

Seleccionando una parte central del sonido podremos probar las opciones.



5

De este modo insertamos un silencio en el sonido.

TRUCO

Si hacemos clic en cualquiera de las ventanas del gráfico del sonido mientras reproducimos, cambiará la forma de representación.

ra situada en la parte inferior de la ventana de audio hasta ver el final de la selección y con el botón derecho pulsado, movemos el ratón hasta ajustar la marca del final. Ya estamos preparados para cortar el silencio. Pulsamos en el icono o con *CRTL + X*. Volvemos a ajustar toda la onda a la ventana y listo (ver fig. 3).

Podemos también seleccionar todo el sonido de una vez con *CTRL + A* y escoger sólo el que vemos en la ventana con *CTRL + W*.

Vamos a aclarar cómo funcionan los controles de "Play". El primer control sólo reproduce lo que está seleccionado una sola vez. Si queremos un mayor control de reproducción tenemos que elegir el icono . En la pestaña "Play" de opciones encontramos todas las posibilidades de reproducción en "User play button". Antes de nada, seleccionemos una parte central del sonido para probar todas estas opciones (ver fig. 4).

- **1. "All".** Reproduce todo desde el principio.
- **2. "Selection".** Reproduce toda la selección
- **3. "Unselected".** Reproduce todo lo que no está seleccionado. Cuando llega a la marca del comienzo salta hasta la marca del final y continúa.
- **4. "View".** Reproduce lo que se ve en la ventana.
- **5. "Finish".** Reproduce tres segundos antes de llegar a la marca final.
- **6. "Intro/Loop/end".** Se reproduce desde el principio, hace un bucle de la selección y sigue hasta el final.
- **7. "Loop".** Si está marcado, repetirá cuantas veces se indique una reproducción de la selección.

Vamos a seguir copiando una selección e insertándola en cualquier parte del sonido.

Primero, hagamos una selección. Para copiar en memoria pulsamos el icono o *CTRL + C*. Para insertar en el comienzo del sonido elegir la opción "Paste at" Beginning" en el menú "Edit" o *CTRL + B*. Para insertar al final *CTRL + F* ("Paste at End") y para insertar en el interior del sonido *CTRL + F* ("Finish market").

También podemos realizar una copia de la memoria, pegando en una nueva ventana, creando así un nuevo sonido. Esta operación la realizamos con la opción "Paste new" (*CTRL + P*) en el menú "Edit".

Hay dos opciones muy interesantes en el menú de edición que son "Mix" y "Replace" .

La opción "Mix" (*CTRL + M*) permite mezclar la copia contenida en memoria con la selección actual. Antes de mezclar, proporciona la opción de hacerlo con más o menos volumen.

La siguiente herramienta es "Replace", que admite reemplazar el trozo de sonido copiado en memoria por el trozo seleccionado en el sonido.

Por último, la opción "Trim"(*CTRL + T*) del menú "Edit" es similar a la función de "recortar" utilizada en diseño gráfico, por ejemplo, por el *Paint Shop Pro*. Borrará todo el sonido que no esté seleccionado, y lo que queda se ajusta a la ventana.

Para finalizar, vamos a insertar un silencio en el sonido de 2 segundos. Este espacio sin audio se insertará después de la marca de comienzo de selección. Colocamos la marca donde queremos hacer el inserto. Elegimos la opción "Insert silence" del menú "Edit" y en la casilla numérica escribimos 2.0 y "OK" (ver fig. 5).

Todavía nos queda mucho que aprender de este programa, pero lo haremos como siempre, en la práctica. Así que ...

En el próximo número...

... empezaremos a realizar los efectos especiales de audio de nuestro juego.

Instrucciones básicas del lenguaje

En el número anterior vimos algunos conceptos básicos de programación, necesarios para entender la filosofía del lenguaje *BASIC* que se esconde tras *Blitz3D* (B3D)

SENTENCIAS

Antes de continuar, es necesario conocer qué son las sentencias. En realidad, cuando ejecutamos un programa, le estamos dando al ordenador un conjunto de órdenes definidas. Estas órdenes se denominan sentencias y las hay de distinto tipo según su funcionalidad:

- 1. Sentencias de flujo de programa, comúnmente llamadas "sentencias condicionales".
- 2. Sentencias para bucles.
- 3. Sentencias para cálculos o de asignación.
- 4. Sentencias de control de programa.

FLUJO DEL PROGRAMA

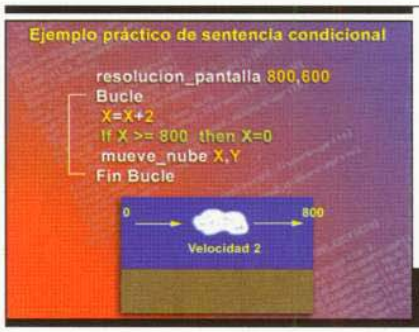
El flujo de un programa es controlado por sentencias condicionales, las cuales permiten cambiar el rumbo de la ejecución del programa después de comprobar y evaluar si se cumple cierta condición. Estas condiciones están formadas por expresiones que contienen operadores de comparación, por ejemplo, una expresión de este tipo podría ser *vida < 1* o *x=100 And y=200*

Los símbolos "<" o "And" son los operadores de comparación y lógicos, y en *B3D* disponemos de los siguientes: < (menor que), > (mayor que), <= (menor o igual que), >= (mayor o igual que), = (igual que), <> (distinto que). En todos estos operandos se necesitan dos valores para comparar. También disponemos de operandos lógicos como: *Not* (compara que no se cumpla una condición), *And* (compara que dos expresiones son ciertas), *Or* (compara que al menos una de las dos expresiones es cierta), *Xor* (compara que sólo una de las dos expresiones es cierta).

Siempre que realicemos estas comprobaciones, obtendremos como resultado los valores 0 (False) o 1 (True). Además, podemos encadenar expresiones condicionales separadas por operadores lógicos y englobar expresiones dentro de otras utilizando paréntesis, por ejemplo:

```
(vida<1 or energia<100)
and puntos=0.
```

En este ejemplo, se evalúa primero lo que está dentro del paréntesis y el resultado se comparará con la siguientes expresión; así, se cumpliría la condición *si puntos vale 0* y *vida es menor que 1* o *si puntos vale 0* y *energía es menor de 100*.

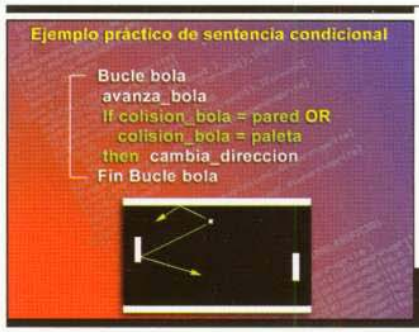


Un esquema del movimiento continuo y circular de una nube en el horizonte. Se actualizan coordenadas por medio de una condición.

Hay que tener en cuenta que los operadores lógicos convierten sus operandos a valores enteros y el resultado que producen también. A continuación, se muestran algunos ejemplos de expresiones condicionales válidas:

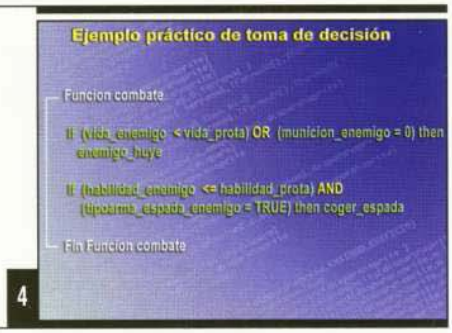
```
coordx_enemigo <> coordx_jugador
And Not disparo=0
```

En esta expresión se comprueba si las coordenadas x del enemigo y del jugador son diferentes y que, por ejemplo, se haya producido un disparo, es decir, negar el valor 0 de disparo es ponerlo a 1; luego la expresión también podría escribirse como *And disparo=1*.



Un ejemplo del uso de un operador lógico OR para el control del rebote de una pelota.

En la figura se muestra una tabla con los tipos de operadores lógicos y condicionales disponibles en Blitz3D.



La combinación de operadores lógicos y de comparación permiten simular cierta inteligencia en un programa.

Con este tipo de sentencias es como hacemos que nuestro programa "piense" y tome decisiones.

Pero para implementar estos controles es preciso disponer de estructuras establecidas para ordenar el flujo de un programa. *B3D* posee varios tipos de estas estructuras condicionales.

ESTRUCTURA IF ... THEN

Su construcción básica es la siguiente:

```
If expresión Then sentencias1
Else sentencias2
```

o también

```
If expresión
  Sentencias1
Else
  Sentencias2
Endif
```

Se evaluará la expresión y si (If) el resultado es verdadero entonces (Then) se ejecutarán las sentencias1; si el resultado es falso, se ejecutarán las sentencias2. Por ejemplo:

```
If vida<1 Then jugador_muere
Else vida=vida-1
```

Si (If) *vida* es menor que 1 entonces (Then) se ejecuta la función "jugador_muere"; si no (Else), se decrementa una vida.

Se pueden encadenar varias estructuras *If*.

```
If expresión1
  Sentencias1
Else If expresion2
      Sentencias2
     Else
      Sentencias3
Endif
```

En esta estructura encadenada, las "Sentencias3" se ejecutarán solamente si la "expresion1" del *If* o la "expresion2" del *Else If* (Elself) fueran verdaderas (True)

ESTRUCTURA SELECT ... CASE

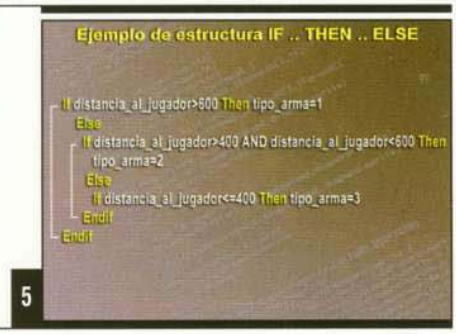
```
Select expresión
  Case expresiones1
  Case expresiones2
  Default expresiones3
End Select
```

Este tipo de estructuras nos ayuda a evitar un gran número de estructuras If ... Then en nuestro programa. Imaginaos que queréis dibujar en pantalla diferentes explosiones dependiendo del tipo de disparo que se haya efectuado. Esto se podría realizar mediante muchos *If* , por ejemplo,

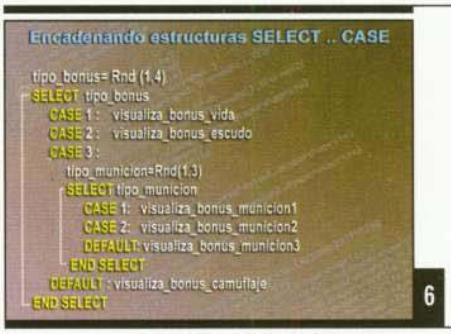
```
If disparo=1 Then
dibujar_explosion1
If disparo=2 Then
dibujar_explosion2
If disparo=3 Then
dibujar_explosion3
```

Realmente este sistema es algo engorroso y poco elegante. La estructura "Select ... Case" es una alternativa más apropiada.

```
Select disparo; según el valor
de disparo
Case 1 Dibujar_explosion1 ;
  en caso de que disparo=1
  dibujamos la explosión 1
Case 2 Dibujar_explosion2 ;
  en caso de que disparo=2
  dibujamos la explosión 2
Default Dibujar_explosion3 ;
  en el caso restante, es
  decir, disparo=3
  dibujamos la explosión 3
End Case
```



Las estructuras IF .. THEN se pueden combinar entre sí.



Las estructuras SELECT.. CASE son fácilmente encadenadas.

SENTENCIAS PARA BUCLES

En este tipo de estructuras se realiza un bucle de las sentencias hasta cumplirse determinada condición.

ESTRUCTURA WHILE ... WEND

```
While expresion
  sentencias
```

Una estructura WHILE ..WEND permite controlar el flujo de la partida.

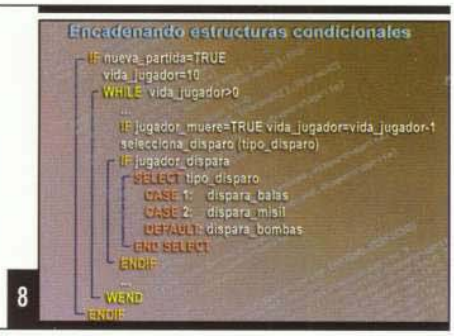
```
Wend
```

Mientras (While) se cumpla la expresión, se ejecutan las sentencias.

Esta estructura es utilizada cuando queremos que se realicen operaciones sólo cuando se cumpla cierta condición. Antes de entrar en la estructura para realizar el grupo de sentencias, se evalúa la expresión y mientras ésta sea verdadera, se permanecerá en el interior de la estructura. En el ejemplo:

```
vida_jugador=3
While vida_jugador>0
 dibuja y mueve jugador
 dibuja y mueve los enemigos
 ataca los enemigos al jugador
 If disparo_enemigo colisiona
 con jugador Then
 vida_jugador=vida_jugador-1
Wend
muere_jugador
```

Asignamos 3 vidas al jugador. Mientras (While) las "vidas del jugador" no sean 0, "dibujaremos y moveremos al jugador y a los enemigos", "atacarán los enemigos". Si un "disparo del enemigo" "alcanza al jugador", éste pierde una vida. Se vuelve a preguntar si la "vida del jugador" no es 0; como ahora vale 2 se vuelve a ejecutar las acciones anteriores y así hasta que las vidas del jugador valgan 0. Se saldrá de la estructura y se ejecutará "muere_jugador".



Se pueden encadenar distintos tipos de estructuras condicionales, según las necesidades del programa.

ESTRUCTURA REPEAT ... UNTIL

```
Repeat
  sentencias
Until expresion
```

Repetir (Repeat) las sentencias hasta (Until) que se cumpla la expresión.

Su misión y funcionamiento es la misma que la estructura "While ...End". La única diferencia es que la evaluación de la expresión se realiza después de ejecutar las sentencias contenidas en la estructura. Se utiliza cuando queremos que el grupo de sentencias se ejecute al menos una sola vez. Vamos a realizar el ejemplo anterior pero con esta estructura:

```
vida_jugador=3
Repeat
 dibuja y mueve jugador
 dibuja y mueve los enemigos
 ataca los enemigos al jugador
 If disparo_enemigo colisiona
 con jugador Then
 vida_jugador=vida_jugador-1
Until vida_jugador=0
muere_jugador
```

ESTRUCTURA REPEAT ... FOREVER

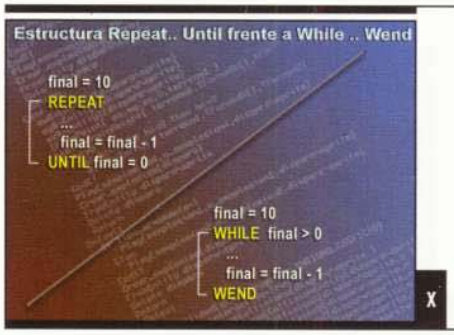
```
Repeat
  Sentencias
Forever
```

Repetir (Repeat) las sentencias para siempre (Forever).

En realidad esta estructura sirve para realizar un bucle infinito.

Si en algún momento deseamos salir de este bucle debemos utilizar la función Exit. Por ejemplo:

```
Repeat
 Print "esto no tiene fin
 hasta que aprietes un tecla"
 If aprietas cualquier tecla
 Then Exit
Forever
```

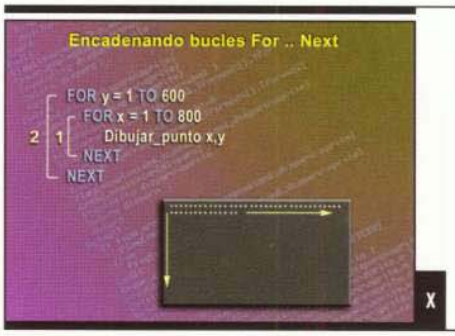


El mismo resultado con dos estructuras de bucles distintas.

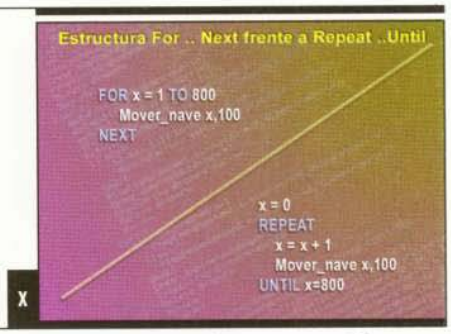
ESTRUCTURA FOR ... NEXT

```
For variable = valor inicial
To valor final Step salto en
el incremento del valor
 sentencias
Next
```

Este tipo de estructuras para bucles funciona preguntando por la evolución del valor de una variable. Antes de entrar en el bucle, se asigna el "valor inicial" a la "variable" y se entra en el bucle. Una vez ejecutadas las "sentencias" con la función "Next" se regresa al *For*. Se incrementa 1 unidad el valor de "variable" (opcionalmente se puede incrementar en las unidades que se indique en "Step") y se vuelve a entrar en el bucle. Esta operación se repetirá hasta que el valor contenido en "variable" alcance el "valor final". Por ejemplo, queremos contar del 1 al 10 de dos en dos e imprimirlo en la pantalla:



En la figura, el bucle interior (1) se ejecuta antes que el exterior (2).



El mismo bucle con dos estructuras diferentes.

```
For contador = 1 to 10 step 2
  print contador
Next
```

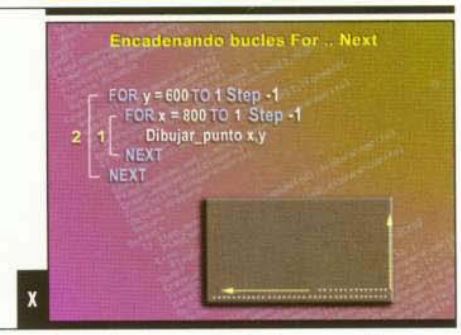
Para realizar la cuenta pero al revés, se deben cambiar el "valor final" por el "valor inicial" y colocar "Step" en negativo. Por ejemplo, si queremos hacer una cuenta atrás del 10 al 1:

```
For contador =10 to 1 Step -1
  print contador
Next
```

También puede haber bucles dentro de otros bucles. El bucle interior se repetirá tantas veces como indique el bucle exterior.

En *B3D* existe un tipo de bucle *For* que nos permite recorrer cada uno de los objetos que forman parte de una estructura de datos. Se trata de la estructura For ... Each

```
For variable = Each nombre de
la estructura de datos o tipo
de datos
```



Se pueden recorrer los bucles en sentido contrario con la opción STEP.

```
  Sentencias
Next
```

Es muy utilizado para actualizar grandes grupos de objetos. Pero explicaremos el uso de esta sentencia con más detalle cuando hablemos de las estructuras de datos.

USO DE MATRICES

En el número anterior explicamos qué eran las matrices o *arrays*. Ahora le daremos algún uso en nuestro programa. Supongamos que queremos guardar información sobre la posición de cuatro llaves dentro de un laberinto en 2D. Estas posiciones vienen determinadas por las coordenadas X e Y. Definimos la matriz: Dim llave (3,1) ="10,20, 20,30, 150,340, 100,400"

La primera llave (la número 0) está en las coordenadas X=10 e Y=20, la segunda en X=20 e Y=30, y así sucesivamente. En el siguiente ejemplo imprimimos la coordenada Y de la llave 3.

```
Dim llave (3,1) ="10,20,
20,30, 150,340, 100,400"
Print llave(3,1)
```

Y en el siguiente, imprimimos las coordenadas X e Y de las cuatro llaves:

```
Dim llave (3,1) ="10,20,
20,30, 150,340, 100,400"
```

Damos valores a la matriz

```
For numero_llave=0 To 3;
Recorremos la tabla
 Print llave (numero_llave,0);
 imprimimos la coordenada X de
 la posición numero_llave
 Print llave (numero_llave,1);
 imprimimos la coordenada Y de
 la posición numero_llave
Next
```

Las matrices pueden resultar útiles para manejar datos con un elemento común, pero si tenemos un grupo de objetos con los mismos parámetros no resulta del todo eficaz.

Un ejemplo de uso de matrices para imprimir un grupo de ovnis.

Supongamos que queremos controlar la posición (Coordenadas X e Y) y los gráficos (3 tipos de gráficos) de una horda de 50 naves enemigas. Se podría utilizar una matriz como Dim enemigos (49,4) y luego recorrerla con un bucle *For ... Next*. Esto requiere mucho esfuerzo por parte del ordenador y del programador. Para solucionar este problema *B3D* ofrece la posibilidad de definir tipos de datos en estructuras llamadas *Type*. El ejemplo anterior se podría definir con este tipo de estructura de la siguiente forma:

```
Type tipo_enemigo;
Definimos un cómo serán los
enemigos.
 Field x,y;
Coordenadas X e Y de cada enemigo
 Field grafico[2];
3 gráficos diferentes por
cada uno de ellos
End Type
Dim enemigos.tipo_enemigo(49);
Creamos 50 enemigos distintos
de la misma clase
For n=0 to 49
 enemigos(n)= New tipo_enemigo;
Inicializamos lo 50 nuevos
enemigos para poder usarlos
Next
```

Realmente, es algo difícil de entender, pero más rápido, eficaz y elegante. Cuando comprendas su funcionamiento, seguramente utilizarás este sistema las veces que puedas en tus programas. No te preocupes, en el siguiente número explicaremos con más detalles este fantástico sistema.

Secuenciadores. Cubase VST (I)

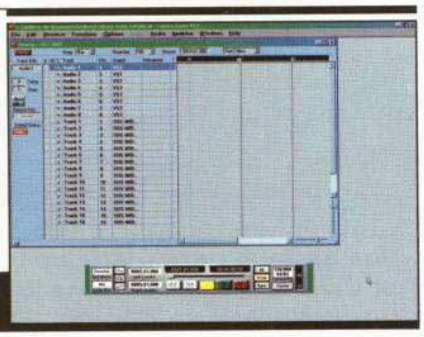
La casa Steinberg se remonta a la época del Atari ST. Desde entonces, ha conservado su filosofía de diseño. Tanto que, si has conocido la primera versión de su secuenciador *Cubase*, no tendrás problema para adaptarte a la última versión, casi veinte años después. (Incluido en el CD-ROM).

La saga de secuenciadores *Cubase* ha sido su producto estrella y ha creado un estándar mundial en la informática musical doméstica y profesional.

Con la llegada de las posibilidades multimedia, Steinberg adaptó su secuenciador *Cubase* a los nuevos tiempos, incluyendo en su gama VST la posibilidad de mezclar pistas de audio con las tradicionales pistas MIDI. Actualmente, es utilizado en estudios de grabación profesionales, en cine, y en cualquier medio audiovisual, y por supuesto por miles de músicos.

LOS PRIMEROS PASOS

Para empezar, debemos saber que *Cubase* es, ante todo, un secuenciador MIDI. No obstante, posee la capacidad de grabar y reproducir pistas de audio y mezclarlas. Para algunos aspectos de este tutorial, sería conveniente que tuvieras conectado al ordenador un instrumento MIDI, como un teclado maestro. *Cubase* es un programa muy extenso y potente, así que en estos tutoriales expondremos una manera rápida y pragmática de utilizarlo. Lo que aprendas aquí puedes aplicarlo a cualquier versión de la gama VST de *Cubase*.



1

Configuración por defecto al arrancar Cubase.

Al ejecutar *Cubase* se carga una configuración básica por defecto, consistente en una ventana de arreglo con 8 pistas de audio y 16 pistas MIDI (ver fig. 1).

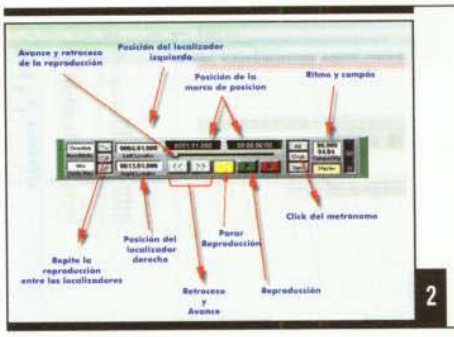
En esta primera entrega conoceremos el uso de los elementos de la interfaz. Para ello, vamos a cargar la canción "def.all" que se encuentra en el directorio "\audio" del CD con la función "Open" del menú "File". Esta canción es un pequeño ejemplo que contiene una pista de audio y ocho pistas MIDI.

Para empezar, presentaremos un elemento muy importante en el *Cubase*: la barra de transporte. Es un panel flotante que contiene las funciones de reproducción, paro, rebobinado, grabación, ajuste de ritmo, etc. En la figura 1 se puede ver un esquema de este panel y una descripción de sus funciones. Es importante que aprendas la equivalencia de los botones de la barra de transporte en el teclado, ya que usar el ratón cuando grabas o manipulas esta paleta es un poco engorroso.

LA BARRA DE TRANSPORTE

Vamos a practicar la reproducción. Sobre los botones de control podemos ver una barra deslizadora que nos permitirá desplazar la marca de posición de la canción igual que utilizamos los botones de "Atrás" y "Adelante" (ver fig. 2).

Iniciamos la reproducción pulsando el botón "Play" o con la tecla "*Intro*" del teclado numérico. Para parar la ejecución pulsamos el botón "Stop" o la barra espaciadora. Para irnos a la posición del localizador izquierdo pulsamos "1" (por defecto, siempre estaremos hablando del teclado numérico, mientras no se indique lo contrario) y para colocarnos en la posición del localizador derecho pulsamos "2". Si lo que deseamos es ir al comienzo de la can-



2

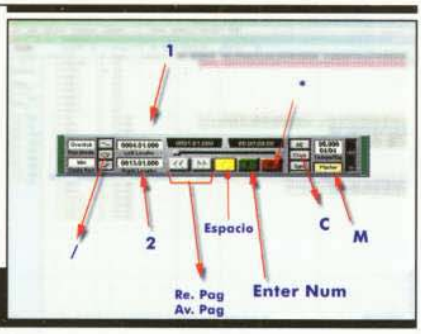
Con la barra de transporte reproducimos las canciones.

NOTA

Básicamente, sólo existen 16 canales MIDI distintos. Esto no significa que estemos limitados a 16 pistas MIDI; es decir, podemos tener 10 pistas diferentes asignadas al canal MIDI 5. Las 10 pistas sonarán con el mismo sonido pero pueden tener distintos arreglos musicales.

TRUCO

Se puede utilizar el teclado numérico para reemplazar a la barra de transporte en cualquier momento, incluso estando en otra ventana de diálogo.



3

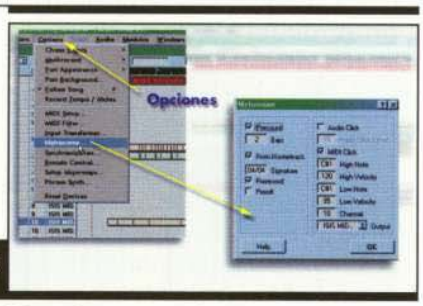
Podemos avanzar o retroceder a nuestro antojo.

ción pulsamos "5" o la tecla "*Inicio*". Para ir al último sitio donde se paró la música pulsamos el "9".

Para correr hacia delante o hacia atrás utilizaremos las teclas retroceder página *RePag* y avanzar página *AvPag* situadas fuera del teclado numérico (ver fig. 3).

A continuación, vamos a controlar la posición de los localizadores. Estos elementos nos sirven para indicar el comienzo y el final de la grabación o de la reproducción cíclica. Para colocar o desplazar el localizador izquierdo, nos situamos en la regla de compases y pulsamos con el botón izquierdo del ratón donde queremos colocarlo; realizaremos lo mismo, pero con el botón derecho, para el localizador derecho. Podemos realizar la misma operación introduciendo los valores en las casillas correspondientes en la barra de transporte, haciendo *clic* dos veces.

Otras funciones de la barra de transporte son: el *clic* de metrónomo, el sincronismo, el ritmo y unidad de tiempo, la reproducción cíclica, el modo de grabación y el tipo de grabación cíclica. No las veremos ahora todas pero sí las más utilizadas.



4

El metrónomo es una función que nos será de gran utilidad.

Una función muy útil es el metrónomo, podemos activarlo pulsando la tecla *C* y configurarlo en la opción "Metronome" en el menú "Options" (ver fig. 4).

Si queremos que se repita la reproducción entre los localizadores, debemos activar la "Reproducción cíclica" con la tecla "/".

Mediante el botón "Master" (tecla "*M*") activamos el tiempo y el compás definidos en lista del "Maestro de pistas" (Mastertracks), que explicaremos en otra ocasión. Si lo tenemos desactivado cambiarán a los valores por defecto.

LA VENTANA DE ARREGLO

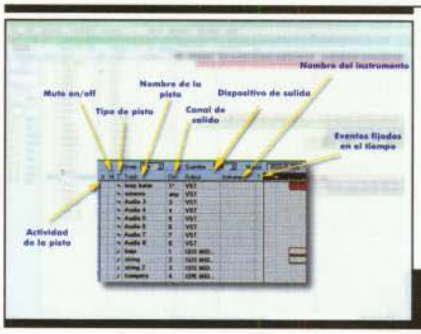
El arreglo está formado por pistas, donde grabamos y reproducimos la canción. El sistema de pistas permite grabar los instrumentos por separado para poder editarlos uno a uno. En la figura 2 se puede ver una descripción de la parte de datos de la ventana de arreglo y en la figura 3 una descripción del visualizador del arreglo. Aquí es donde vemos la música grabada en cada pista y donde la editaremos.

En primer lugar, vamos a configurar la forma de visualizar cada pista. Ahora podemos observar el nombre de pista y observaremos que cada una tiene un color distinto. Si queremos comprobar qué contiene cada una de una forma indicativa, elegimos la opción "Show events" en la opción "Part Appearance" del menú "Options".

Si deseamos tener más espacio para el visualizador de pistas, podemos ocultar la parte de información de pistas pulsando el botoncito aislado situado debajo de "Part Info", en la parte inferior izquierda de la ventana de arreglo.

Podemos observar que cada pista lleva asociada 8 casillas (ver fig. 5):

- **1. Casilla "A":** Indica la actividad de la pista.
- **2. Casilla "M":** Si la pista está mutada o no.
- **3. Casilla "C":** Indica el tipo de pista. Haciendo *clic* saldrá la lista con los distintos tipos.



5

Cada pista lleva asociada ocho casillas.

- **4. Casilla "Track":** Haciendo *clic* dos veces podemos asignar un nombre a la pista.
- **5. Casilla "Chn":** Indica el número de canal para la pista: 8 pistas mono para audio o 4 estéreo y 16 pistas MIDI.
- **6. Casilla "Output":** Aquí le indicamos al *Cubase* por qué dispositivo sonará la pista; la tarjeta de sonido (SB, ISIS) depende de la tarjeta o instrumento externo (Maestro, MPU o Módem, dependiendo de cómo tengas instalado el equipo y en qué interfaz esté conectado el instrumento)
- **7. Casilla "Instrument":** Sólo se activa en las pistas MIDI y sirve para anotar qué nombre de instrumento tiene asignado la pista. Se activa haciendo doble *clic*.
- **8. Casilla "T":** Si se activa, indicará que esa pista tiene los eventos fijados en el tiempo. Generalmente debe estar desactivada.

Con los cursores podemos desplazarnos por las pistas hacia arriba o hacia abajo. Si queremos que sólo suene la pista seleccionada pulsamos la tecla "*S*"; al desplazarnos, la pista activa seleccionada será la que suene sola y las demás se mutarán.

En el próximo número...

... aprenderemos los métodos básicos para manejar el *Cubase* y el uso de las herramientas de edición, así como a trabajar en la ventana de arreglo.

ZX Spectrum (I) De 1980 a 1984

¿Qué podemos decir de este ordenador?. Su creador, Sir Clive Sinclair, lanzó al mercado su primer Spectrum -el ZX-80- en 1980.

Estaba basado en el primer procesador de Zilog Z80, poseía la "enorme" cantidad de 1Kb de memoria y era monocolor. Fue destinado mayoritariamente a la enseñanza, ya que utilizaba el lenguaje *BASIC*. Sinclair Research Limited nació a raíz de comercializar el ZX-81, de iguales características que su predecesor, menor tamaño y teclas sensibles al tacto. El gran acierto de Sinclair fue situar sus precios por debajo de los de Atari o Commodore. Precisamente, la salida del fantástico Commodore VIC-20 obligó a Sinclair a lanzar el primer ordenador doméstico de ventas masivas en Europa, el ZX Spectrum de 16 K y 48 K, con un sistema de teclas de goma multifunción un poco particular y a veces odiosas. Aun así, la programación en código máquina de su microprocesador era relativamente fácil y cualquiera podría desarrollar juegos para esta máquina. Esta particularidad fue el trampolín que catapultó al ZX-Spectrum a la cima durante cerca de 10 años, ya que emergieron infinidad de casas desarrolladoras de juegos que producían una cantidad ingente de títulos diversos que las distribuidoras vendían a precios asequibles, inundando un mercado que devoraba y devoraba Spectrums.

La historia moderna que conocemos del mundo de los videojuegos la han forjado compañías y juegos que nacieron en la época de los 8 bits. Desde que hizo su aparición en Inglaterra el ZX 80 de Sir Clive Sinclair, los juegos para ordenador han adquirido otra dimensión en diversidad de géneros y tecnología. Sin embargo, no todo estaba inventado aún; en estos años los géneros que dominaban eran juegos de aventuras, arcades, matamarcianos, juegos deportivos o aventuras conversasionales. La tecnología de entonces no permitía un gran desarrollo tecnológico y el usuario estaba acostumbrado a estos géneros.

La llegada de los procesadores de 16 bits, principalmente al mundo de los PC de IBM, abrió la puerta a nuevas posibilidades técnicas, lo que contribuyó al desarrollo y evolución de nuevos géneros.

AVANCES DE UNA DÉCADA

Sin más dilación, vamos a exponer los avances técnicos que sufrieron los juegos para ZX Spectrum a lo largo de su abultada historia. Mencionaremos sólo los títulos que han destacado por su tecnología.

1982

El Spectrum empezaba a conocerse en los hogares; todavía su programación y posibilidades eran desconocidas y comenzaban a aparecer juegos que versionaban las recreativas del momento como *Galaxian* o *Scramble* para promocionar la máquina. Quizás fue la adaptación del *Hobbit* el primer juego famoso para Spectrum. Nació por entonces la primera compañía de videojuegos importante, que se dio a conocer con el matamarcianos *Arkadia*. Nos estamos refiriendo a Imagine. En este año salió al mercado el primer simulador de vuelo para Spectrum,

1

Multitud de juegos de Spectrum se versioneaban para Amstrad.

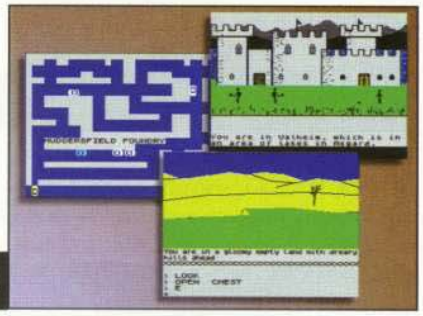
2

El ordenador de 8 bits más popular en Europa.

NOTA

¿Sabías que *Knight Lore* (1985) de Ultimate se programó antes que *SabreWulf* (1984)? Debido a su nueva tecnología y calidad, *Knight Lore* fue reservado para no ensombrecer el lanzamiento de *SabreWulf* (350.000 copias vendidas).

3 *Dos fotos del ZX Spectrum, los ordenadores de 8 bits más populares de los años 80.*



4 *Unas capturas de los primeros juegos para Spectrum del año 1982.*

DEFINICIÓN

▸ SPRITE EN EL SPECTRUM

Es un gráfico gestionado por una rutina especial que se puede mover por la pantalla sin alterar el fondo, dando la sensación de estar pasando por encima del mismo.

DEFINICIÓN

▸ TÉCNICA FILMATION

Esta técnica permitía realizar juegos en perspectiva isométrica en un Spectrum. Funcionaba dibujando las habitaciones en la pantalla en perspectiva isométrica, y por medio de sprites diseñados en diferentes ángulos daba la sensación de un entorno 3D. Después se desarrolló la técnica filmation 2, que consistía en mover los escenarios mediante scroll en vez de los personajes.

Flight Simulation de Psion; versión del primer *Flight Simulator* en blanco y negro del ordenador TRS-80 que posteriormente adquiriera Bill Gates para su serie de simuladores de vuelo.

1983

A partir de ese momento, prácticamente detrás de cada buen juego nacía una nueva compañía. Cabe destacar algunas como Incentive, Ocean o Bug Byte. Esta última desarrolló *Manic Miner*, quizás el primer juego de plataforma con niveles estudiados al milímetro, descubriendo un nuevo tipo de jugabilidad que pronto adquirirían multitud de desarrolladoras para sus juegos. También este año hace su aparición en escena la compañía más innovadora, sentando precedente en la creación de juegos para Spectrum: Ultimate Play The Game. Fueron sus miembros, sin duda, los creadores de los arcades de aventuras más fantásticos de la historia de los 8 bits. Títulos como *Atic-Atac* (en perspectiva cenital), *Sabre Wulf*, *Jet Pac* o *Pssst*, definieron el género, con gráficos detallados al máximo, montones de niveles y sin duda acción, mucha acción y en tan sólo 16 Kb de memoria.

1984

A partir de este año se inicia el *boom* del software para Spectrum. Las ventas decaían por la falta de juegos originales y las casas se tuvieron que poner las pilas. El género arcade shoot'em ups dominaba el mercado, pero el usuario pedía más calidad y nuevos géneros. La programación del Z80 ya no era un secreto y el mercado estaba inundado de libros y revistas especializadas. Esto contribuyó a explotar el rendimiento de los ZX Spectrum al máximo. Fue un año de muchos avances técnicos, y aún sorprenden los nuevos lanzamientos que parecían imposibles para el Spectrum. La calidad gráfica del Spectrum era tosca y el sonido se limitaba a un montón de pitidos monofónicos a veces molestos. Los colores en el Spectrum eran torpes de manejar, sólo se admitían 16 colores diferentes a la vez y 1 color por cada 8 píxels. Hasta la llegada de técnicas como la *Filmation* o la técnica para gestionar *sprites* no se pasaba de un mismo color para el fondo o de la misma perspectiva.

A finales de ese mismo año surge otra maravilla de Ultimate, *Knight Lore*. Primer juego que utilizó la técnica *Filmation* y que revolucionó el concepto de juegos para Spectrum. No tardaron mucho las demás desarrolladoras en engancharse al carro de la isometría.

Pero esto no termina aquí. Un escocés llamado Sandy White vuelve a sorprender a la comunidad del Spectrum con un juego que también utilizaba la perspectiva isométrica pero generada en 3D, *Ant Attack*. Después, desarrolló *I,of the mask* que utilizaba 3D real con objetos sólidos. Abundaban los simuladores espaciales en 3D con gráficos vectoriales como *3D Starstrike*, *Starion* o *Codename Mat*.

En este año también surgieron juegos que han hecho historia como *Avalon* de Hewson Consultant o la continuación del ya clásico *Manic Miner*, *Jet Set Willy* de Matthew Smith.

El género deportivo tuvo su mayor exponente en el que quizás sea el padre de todos los buenos juegos de tenis, sin lugar a dudas, el *Match Point* de Psion, que triunfó por tener unos gráficos simples pero cargados de realismo.

No queremos dejar este magnífico año sin hablar del juego que trajo el género *beat'em up* (peleas) a los ordenadores domésticos: nos estamos refiriendo al *Kung Fu* de Bug Byte.

En el próximo número...

... continuaremos este paseo por la historia de esta gran maravilla que fue el ZX Spectrum y que tan buenos ratos nos hizo pasar.

Cuestionario Videojuegos 4

Preguntas

1. Enumera las estructuras condicionales del Blitz3D para manipular el flujo de un programa.
2. Enumera las distintas estructuras para bucles del Blitz3D.
3. ¿Qué se entiende por protagonista principal de un juego?
4. ¿Qué aspectos de los personajes del juego tenemos que diseñar con detalle antes de modelarlos, animarlos y programarlos?
5. En el Milkshape3D enumera las teclas asociadas a las funciones de: seleccionar, mover, rotar y escalar.
6. Describe el procedimiento para reemplazar el color de una imagen en Paint Shop Pro.
7. En el programa GoldWave, ¿cómo podemos reproducir una selección tres veces seguidas?
8. ¿Cómo se selecciona un trozo del sonido en el GolWave?
9. ¿Qué significan las letras "M", "C" y las palabras "Chn" que aparecen en cada pista en la ventana de visualización de pistas del CUBASE VST?
10. Enumera las teclas asociadas a las siguientes funciones: reproducir, parar reproducción, ir a la posición del localizador izquierdo, ir a la posición del localizador derecho, ir al principio de la canción.

Respuestas al cuestionario 3

▷ **1.** Valores numéricos enteros, flotantes y valores alfanuméricos.

▷ **2.** **A.** Dim edificio (9,19)
B. Habitación = edificio (2,14)

▷ **3.** En un guión técnico se describen las facetas gráficas, sonoras y programación.

▷ **4.** Implementar un comportamiento imprevisible y con competitividad.

▷ **5.** Consiste en formar las caras del modelo utilizando triángulos, los cuales creamos previamente uniendo tres vértices.

▷ **6.** Es un visualizador previo de las imágenes contenidas en un directorio. Se activa con CTRL + B o eligiendo la opción "Browse" en el menú "File".

▷ **7.** Son aparatos que capturan sonidos externos y los transforman en audio digital para poder procesarlos.

▷ **8.** Pro Tools para Macintosh, Cubase para Atari y PC y AnvilStudio para PC.

▷ **9.** Hay que dibujar primero un cubo en el mapa y posteriormente elegiremos la geometría que queramos como un cilindro.

▷ **10.** Dibujamos un cubo que contendrá la luz. Una vez hecho esto pulsamos el botón derecho del ratón sobre la parrilla y elegimos la opción "Light".

Contenido CD-ROM 4

▶ AUDIO

■ Cubase VST
Nueva oportunidad para probar el magnífico programa de edición, con el cual podrás manipular tanto MIDI como pistas de audio.

■ Ashampoo AudioCD MP3 Studio 2000 Morpheus Edition 2.36
Programa para grabar cds y así salvar periódicamente tus composiciones para el juego.

■ Virtual Piano 3.0

Simulación de más de 600 instrumentos y otros interesantes efectos de sonido.

■ RioPort Audio Manager 3.5
Para buscar, grabar, y reproducir fácilmente toda clase de música digital.

■ Crakbone 1.1
Original herramienta con la que podrás convertir mágicamente palabras en melodías.

■ MySoundStudio 2.0
Aplicación muy completa con la cual crear, editar, y arreglar toda clase de música y efectos de sonido.

▶ DISEÑO 2D

■ Ulead Photo Explorer 7.0 Pro
Con este programa podrás organizar de un modo sistemático todas tus imágenes y contenidos multimedia.

■ FotoTime FotoAlbum 3.1.3

Otro excelente programa para organizar nuestras imágenes, cosa que deberemos hacer con eficiencia para controlar bien el diseño del juego.

■ Procreate Painter 7
Demo de la popular aplicación en su versión para Windows, con la cual podrás crear espectaculares ilustraciones y efectos.

■ Groone's KwikPik 2.11
Versátil herramienta para capturar pantallas de un modo rápido y eficiente.

■ XnView 1.21
Para ver, editar y convertir imágenes muy rápidamente podemos usar esta sencilla herramienta.

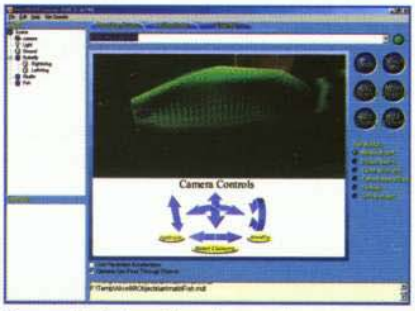
▶ DISEÑO 3D

■ Ulead Cool 3D 3.5
Con esta excelente herramienta podrás realizar títulos en 3D para luego utilizarlos en los créditos del juego.

■ Rhino 3D 2.0
Fantástico editor que permite crear cualquier tipo de objeto en tres dimensiones.

■ Amapi 3D Modeler 6.1
Otro potente editor interactivo para crear modelos 3D de un modo muy profesional.

■ Alice 99 2.01.005



Desarrollo interactivo de mundos enteros en tres dimensiones.

■ DOGA-L1 9905
Especial para principiantes, esta aplicación nos enseña poco a poco a crear objetos en 3D.

▶ PROGRAMACIÓN

■ InterAx 1.2.0
Crea tus propios juegos de aventura en primera persona. Especial para no programadores.

■ VorteX 3.1
Programa juegos en RPG usando generadores de código, un depurador de código, y más...

■ Gmax 1.1
Completa plataforma que te permitirá

programar juegos en 3D. Contiene un montón de útiles herramientas.

■ Suneido 0.9.020313
Interesante programa que te permitirá mantener todas tus aplicaciones.

■ GOLD Parser Builder 1.0.20a
Potente herramienta que puede ser usada con numerosos lenguajes distintos de programación.

▶ JUEGOS Y EMULADORES

Emuladores:

■ Atari Classic Arcade
Nuevo emulador de Atari que te permitirá jugar a seis de los mejores juegos de tipo arcade en tu PC.

■ Amiga para PC
Especial para nostálgicos. Diviértete con este emulador que sin duda te enganchará.

Juegos:

■ Hyperspace Delivery Boy 1.0
Explora el universo y lucha contra monstruos y aliens en este entretenido juego de acción.

■ TankTime 3D 2.5

Excelentes gráficos 3D y sonidos en un juego que sin duda te cautivará.

■ Zone of fighters
Segunda oportunidad para obtener la versión mejorada de nuestro juego.

▶ VÍDEO

■ VScreen 1.0
Software para capturar vídeo y grabarlo en formato AVI.

■ Video Timecode Calculator 1.62
Potente complemento a los editores de vídeo, que te permitirá calcular el código de tiempo.

■ PowerDirector Pro 2.1

Captura y edita vídeo sin pérdida de calidad, con sencillez y rápidamente.

Diseño de los elementos de un juego III: el entorno

En el número anterior estudiamos a fondo todos los aspectos del diseño de la bionave de combate de nuestro protagonista y de todos los personajes que intervienen en el juego.

Toca ahora definir todo el aspecto que tendrán las zonas de combate, como la ambientación y el decorado. Un entorno adecuado hará que el argumento de nuestro juego adquiera consistencia y credibilidad. En "Zone of Fighters" nuestra intención es sumergir al jugador en un ambiente de continuo acoso creado por los numerosos peligros que habitan la zona, por elementos añadidos al torneo (como artefactos voladores cazadores) y por otros luchadores. Además, debemos mantener una sensación de confinamiento ya que, como sabemos, los combates tienen lugar en grandes terrarios.

(■) LA AMBIENTACIÓN

Las batallas se celebran en una superficie cuadrada acotada por las paredes de la urna que cubre el terrario. Esta cubierta proporcionará la sensación de confinamiento deseada y ofrecerá también una perfecta impresión del tamaño de la zona de lucha. Aun así, se aplicarán ciertos elementos atmosféricos como niebla, nieve o la posibilidad de crear el día y la noche. Debes recordar que una ambientación adecuada suma puntos al juego y crea interés sin haber jugado aún con él.

1

En la imagen se puede apreciar un aspecto del terreno de combate cubierto por la urna.

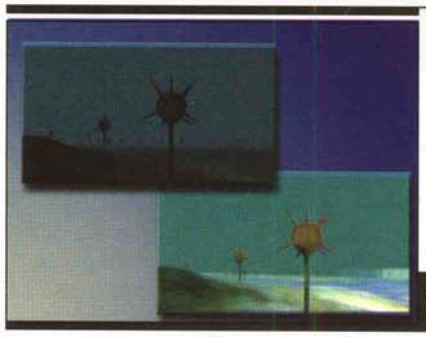
(■) LA LUZ

Siempre que se crea un entorno es necesario aplicar una luz ambiental. Cambiando su color e intensidad añadimos más vida al escenario. Podemos, por ejemplo, simular un atardecer o la oscura noche. En todo caso, es un factor que hemos de tener en cuenta, muy fácil de implementar y con buenos resultados. Lo más normal es crear una luz ambiental blanca con más o menos intensidad. La luz blanca permite que todos los objetos sean vistos con su verdadero color y textura con más o menos claridad. Si, pongamos el caso, creamos una luz roja como ambiental, habremos conseguido nuestro ambiente caluroso, pero perderemos la noción del color real de las cosas; es lo mismo que si te miraras al espejo con una luz roja delante, tu piel pierde su color natural. Para conseguir no perder los colores y crear una ambientación coloreada se utiliza la niebla.

(■) NIEBLA

Haremos hincapié en la utilización de niebla como elemento de sorpresa y de misterio. Creará más peligro y el jugador no dejará de preguntarse qué habrá más adelante. Sin embargo, obtendremos también una cualidad técnica que nos ayudará a aumentar el rendimiento del juego, pudiendo acotar el rango de visualización de la cámara y situarlo detrás de la niebla.

El uso de niebla también ayuda a suavizar los bordes de los polígonos, aunque realmente no sea así. Actúa como filtro de desenfoque: hace que veamos



2

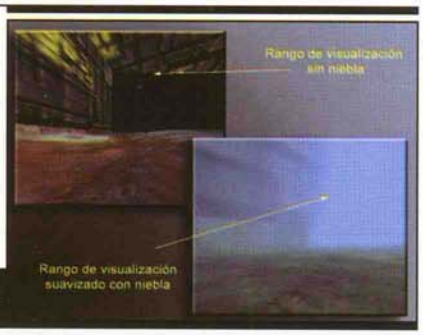
Dos imágenes que muestran el cambio del entorno según la intensidad de la luz ambiental.

NOTA

El rango de visualización es la distancia mínima y máxima que se puede alcanzar a ver con una cámara. Es decir, si situamos un rango máximo de 3000, los objetos situados a partir de ahí no serán visibles. Esto provoca un feo efecto de corte en la imagen. Para evitar esta situación, un buen truco es colocar el rango de la niebla o su distancia un poco antes que el de la cámara, así dicho corte visual se disimulará.

3

La implementación de niebla en un entorno crea misterio e incertidumbre y ayuda a suavizar contornos.



4

La niebla colocada antes del límite de visualización de la cámara oculta el corte en la imagen.

los objetos más difuminados y oscuros dependiendo de su distancia a la cámara. Cambiar su color dará el toque que queremos a la ambientación.

EL CIELO

Este elemento atmosférico es ideal para dar sensación de exterior o *landscape*. Realmente no es nuestro caso, ya que el terreno de juego se halla entre cuatro paredes. Sin embargo, con su utilización creamos un microclima o micro ambiente. El inconveniente de su uso es que eliminaríamos la sensación de encerramiento que buscamos, ya que disimulará más el tamaño real de la urna, y más concretamente su altura, por lo que el jugador parecerá moverse por una enorme superficie sin fin. Así que podemos añadir también como opción el que aparezca o no un cielo en la partida. En su implementación utilizaremos un procedimiento muy común en entornos con cielo: el aplicarle movimiento. Generalmente, es de forma cíclica y en una dirección. Parecerá que hay nubes que desplaza el viento y ganaremos un ambiente con más movilidad.

5

La creación de cielo produce una buen ambientación en el entorno.

OTROS ELEMENTOS ATMOSFÉRICOS

Para reforzar la creación de climas o microclimas en un entorno es muy normal establecer lluvia, nieve o efectos eléctricos como rayos o relámpagos. El uso de estos sistemas puede mermar el rendimiento del juego, ya que se debe dibujar en pantalla simultáneamente gran cantidad de objetos o imágenes (sprites). En la práctica se suele dejar estas características como una opción en el menú, a no ser que la temática o guión del juego lo exija.

En "Zone of Fighters" no habrá nieve ni lluvia pero sí haremos pruebas de cómo quedarían estos efectos implementados, así que dejaremos a vuestro criterio su utilización.

EL TERRENO DE COMBATE

No es necesario comentar la importancia que en este juego tiene el diseño del terreno. Es aquí donde se desarrollan físicamente los combates y es sobre el que descansan todos los demás elementos. En términos de resolución, definiremos su tamaño a 15.000 puntos de lado y una elevación entre 300 y 400 puntos. La definición de esta superficie dependerá de la cantidad de polígonos que la forman, y su aspecto final de las texturas que la cubran. Todos estos aspectos son necesarios tenerlos bien definidos en el diseño, porque pueden influir en gran medida en el rendimiento total del juego. Configurándolos correctamente se puede alcanzar una buena optimización con un resultado más que aceptable.

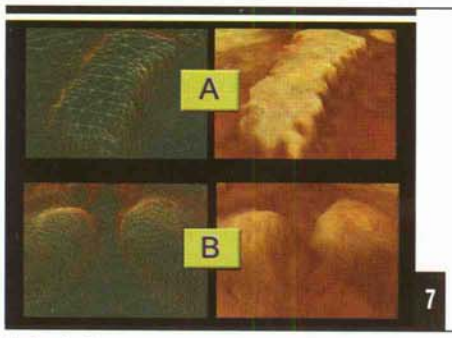
Además, vamos a proporcionar a los terrenos un sombreado en tiempo real o *shading* para conseguir una mayor definición y distinción de perfiles de altura, importantísimo para cuando la cámara esté en posición elevada o con vista cenital. Como este sistema consume algo de recursos, lo incluiremos en el menú como opción.

6

Utilizar un sistema de partículas para crear efectos atmosféricos como la nieve garantiza un aspecto soberbio al entorno.

Jugando con la cantidad de elevación conseguiremos más o menos diferencia entre montes y valles y, como veremos en la creación de los mapas de altura, es necesario saber este dato.

A pesar de todos estos detalles (cantidad de polígonos o sombreado) no conseguiremos un terreno cercano a la realidad sin la utilización adecuada de texturas.

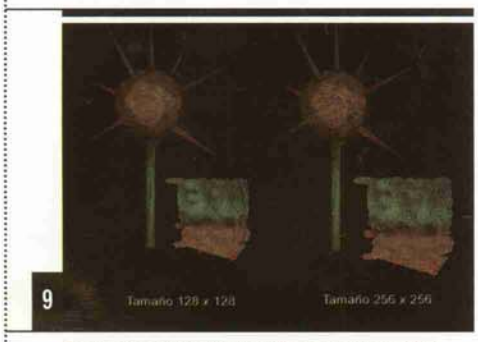


7

En las imágenes se muestran terrenos formados por una cantidad baja (A) y alta (B) de polígonos.

8

Tres ejemplos de aplicación de sombreado (shading) en terrenos.



El tamaño de la textura influye en la calidad visual del objeto y en el rendimiento del juego.

Debemos saber que una textura pierde calidad visual al aumentar su tamaño original. Esto puede suceder cuando queremos cubrir un objeto de tamaño 512 con una textura de tamaño 256. Claro está que podemos entonces hacer la textura del mismo tamaño, es decir, 512. Esto es una buena solución. Pero si tenemos, por ejemplo, un objeto, que en nuestro caso es un terreno con 15.000 puntos de tamaño, no lo sería tanto, ya que una textura de tamaño 15.000 consumiría muchísima memoria y restaría velocidad al juego. Así que lo mejor es diseñar una buena textura con un tamaño aceptable (128, 256 o 512) y utilizar la técnica del *tileado*, es decir, colocar estas texturas repetitivamente sobre el terreno hasta cubrirlo en su totalidad. Hay que tener cuidado a la hora de dibujar los bordes o límites de las texturas para disimular lo mejor posible su repetición.

Sin embargo, a pesar de que nuestras texturas estén perfectamente dibujadas, llega un momento que el tileado queda insultantemente al descubierto, sobre todo cuando la cámara está elevada. Una buena opción para disimular este problema es mezclar texturas. Este sistema dependerá muchas veces de las posibilidades de la tarjeta de vídeo, aunque no tendremos muchos problemas si sólo mezclamos dos. Por ello utilizaremos una textura con el dibujo global de nuestro terreno como piel que lo cubrirá en su totalidad. Evidentemente, quedará demasiado estirada, pero mezclándola con un tileado de otra textura más pequeña se conseguirá un bonito aspecto, incluso visto desde arriba.

En el dibujo de la única textura que servirá de piel podemos definir las partes de nuestro terreno, como zonas verdes, volcanes o arena, y su realización está estrechamente relacionada con la definición de los mapas de alturas que veremos con más detalle en próximos números.

Resumiendo, sólo con cambiar el aspecto de esta textura principal habremos obtenido una nueva ambientación sin cambiar de terreno.

Para "Zone of Fighters" podemos crear uno o dos tipos distintos de terrenos. Por el momento, diseñaremos sólo uno, que constará de partes bien diferenciadas por tonos de color. El terreno estará dividido por un río que lo atraviesa situado en un estrecho valle. En una de las partes encontramos zonas con tonos de color verde y amarillento y en la otra predominan tonos rojizos.

Nos interesa que existan grandes superficies llanas para facilitar el combate, por eso, las pocas elevaciones que hay se reparten en unos volcanes humeantes –que dan un poco de colorido– y en un pequeño desfiladero.

Ya tenemos definido cómo será el terreno de combate y su ambientación, pero nos queda cubrir la superficie con objetos como edificios, árboles, rocas, etc., para decorar un poco el aspecto del juego.

RECORDATORIO

▸ TILE

Un *tile* es una imagen o bitmap que forma parte de un fondo o imagen mayor. Cuando colocamos un grupo sucesivo de tiles estamos "tileando".

(■) EL DECORADO

Estos objetos ayudarán a crear un ambiente menos monótono y definirán las distintas partes en que se divide el terreno; a su vez, contribuirán a formar defensas y escondites para el jugador.

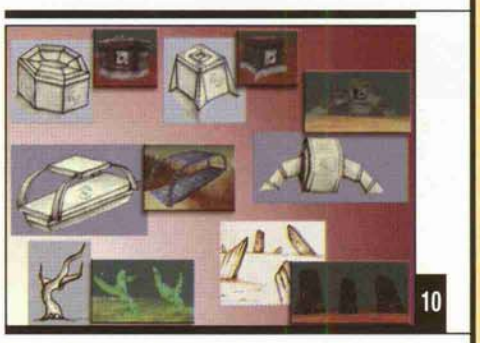
Dividiremos el decorado en varios tipos: edificaciones, objetos orgánicos inanimados y otras decoraciones.

(■) EDIFICACIONES

En este grupo incluiremos elementos arquitectónicos como almacenes, puentes y generadores de corriente. Además, y debido a su carácter pétreo, sumaremos también grandes monolitos de rocas.

Habrá almacenes con dos formas distintas, unos serán cuadrados y otros tendrán forma de pentágono. Estarán construidos con materiales de piedra para que su aspecto esté más a tono con lo que les rodea. Los generadores serán siempre de enorme tamaño, de esta forma inspirarán majestuosidad y diversificarán la línea del horizonte con su silueta. El jugador podrá pasar por debajo de su estructura y servirán para cubrir hondonadas y desfiladeros. Su textura será de un metal con algo de brillo pero muy sucio y con un color próximo al óxido en muchas de sus partes. Y por último, los puentes. Generalmente, se utilizan para salvar el tramo del río en algunos puntos del mapa. Están construidos en metal azul y piedra, y la estructura del suelo posee cuatro arcos que soportan un techo.

Para dar un toque de distinción, a todas las piezas arquitectónicas se les colocará una chapa



En las imágenes se muestran bocetos de diseño y el resultado final.

con el emblema del torneo en una parte visible y de tamaño considerable. Para conservar la estructura de la zona de combate, estos elementos no podrán ser destruidos, pero sí podremos dejar marcas producidas por los impactos de disparos; esto dará más realismo a la acción y proporcionará una visión al jugador de la dureza de las construcciones.

No podemos olvidar las rocas. Las englobamos en el grupo de las edificaciones porque tienen las mismas características. No pueden ser destruidas y los impactos dejan marcas en su superficie. Son de material granítico, de color oscuro y gran tamaño y servirán de escondite y protección en el combate.

(■) OBJETOS ORGÁNICOS INANIMADOS Y OTRAS DECORACIONES

Consideramos objetos orgánicos inanimados a los árboles. Constituyen un bonito adorno para colocar en zonas verdes y acompañar a las plantas carnívoras (dreecks). Para que estén en concordancia con un entorno artificial y devastado podemos diseñar unos árboles sin hojas, con un aspecto pétreo y con tonos de color verde. Además de conseguir una perfecta integración en el ambiente, simplificamos polígonos y formas, lo que ayudará al rendimiento final del juego.

En "otras decoraciones" se pueden incluir todos aquellos aspectos decorativos que contribuyen a crear un entorno adecuado. Por ejemplo, si definimos en el terreno volcanes o fumarolas, estos elementos podrían ser el humo o las llamas. O si creamos un lago cubierto de agua, el reflejo de ésta contribuiría a ambientar aún más.

11

Un ejemplo de utilización de un sistema de partículas para crear volcanes.

En nuestro caso, diseñaremos un sistema de partículas para crear las llamas de los volcanes y proporcionaremos la posibilidad en el menú de aplicar reflejo al agua ya que es el efecto que más velocidad consume.

(■) OBJETOS ESPECIALES

A este grupo pertenecen todos aquellos objetos que participan activamente en el juego y forman parte del uso de los jugadores en cada partida. Los llamaremos *ítems* y son los bonos de puntos, vida, munición, etc.

En nuestro juego, los ítems están repartidos por todo el terreno de juego de forma aleatoria y limitados en cantidad, por lo que sólo puede haber un máximo de diez simultáneamente en el terreno. Todos los jugadores, tanto los humanos como los ovnis controlados por el ordenador, pueden recoger ítems. En el momento en que uno de ellos es capturado, aparecerá uno nuevo en otro lugar del mapa.

Este sistema contribuirá a que el jugador administre sus recursos y dedique un poco de tiempo a la estrategia en el uso y búsqueda de los mismos.

Estos elementos tienen forma cúbica y en su textura identificamos su contenido con un símbolo. Para que puedan ser visibles a cierta distancia, les aplicaremos un emisor de partículas que simule un chorro vertical de luces saliendo del cubo.

A continuación, mostramos la relación de ítems que el jugador puede encontrar y cuyos símbolos mostramos en el cuadro "ítems":

a) Munición:

- 50 unidades de munición de bajo calibre.
- 10 unidades de misiles.
- 6 unidades de bombas de minifusión.
- 6 unidades de bombas de retardo.

12

Cuadro de ítems: objetos que participan activamente en el juego.

b) Extras:

- 100 puntos extras.
- 50 unidades de energía vital.
- 50 unidades de tiempo extra para el camuflaje.
- 100 unidades de tiempo extra para el escudo.

Hasta aquí, hemos definido todos los elementos que compondrán el entorno de nuestro juego. Su diseño abierto permitirá cualquier tipo de ampliación sin demasiadas modificaciones. Podemos añadir más edificaciones, diseñar más animales o plantas, implementar más efectos especiales decorativos y atmosféricos, y, por supuesto, añadir más tipos de ítems. "Zone of Fighters", ante todo, es un juego arcade en donde la temática es pegar tiros y sobrevivir sin muchos calentamientos de cabeza y demostrando la habilidad del jugador. Todo lo que se le quiera añadir aumentará más la diversidad de su contenido pero el argumento seguirá siendo el mismo. Aun así, añadir elementos a su diseño depende de muchos factores, como el tiempo que se quiera emplear en la realización del juego o la variedad de su aspecto. Pero lo más importante es que estos factores nos ayudarán a poner a prueba la creatividad.

En el próximo número...

... terminaremos de analizar los elementos que componen un juego, planteando los tutoriales, diseñando la presentación y los menús.

La pantalla de presentación (I): logotipo del juego

En el número anterior empezamos a conocer las herramientas de Paint Shop Pro. Completaremos el aprendizaje de una manera práctica, y qué mejor forma que realizando el logotipo de nuestro juego.

El emblema que identifica a "Zone of Fighters" es sencillo, de diseño contundente y fácil de memorizar. Estará presente en multitud de partes del juego: en la pantalla de presentación, en el menú, en la textura de la bionave y en algunos elementos del decorado.

Básicamente, está compuesto por dos formas geométricas curvas que rodean a un cuadrado, con diferentes texturas y tratamiento gráfico.

LOGOTIPO. DISEÑO DE FORMAS

Aunque el juego pueda inicializarse en diferentes modos gráficos, la pantalla de presentación aparece con un tamaño fijo de 800 x 600 puntos. Por lo tanto, empezaremos creando una imagen vacía con estas dimensiones, 72 de resolución y con fondo blanco (CTRL + N).

Activamos la barra de herramientas (Toolbars). A continuación hacemos visible el Grid o cuadrícula en "View / Grid" (CTR+ALT+G) y ajustamos su tamaño a 10 unidades en "View / Change Grid and Guide Properties". La cuadrícula nos servirá para centrar y ajustar los polígonos que dibujemos (Fig. 1).

Nuestra primera geometría será el cuadrado del centro del logotipo. Para dibujarlo seleccionamos de la barra de herramientas "Preset Shapes" y elegimos el rectángulo de la librería. Aprovechando la cuadrícula, dibujamos un cuadrado de color negro de 16 cuadros de lado. Ya tenemos un cuadrado vectorial que podemos editar con el selector de objetos (Object Selector), la herramienta situada a continuación de "Preset Shapes". Para centrar el cuadrado en la pantalla pulsamos en el primer icono de "Canvas (Can.)" situado en la pestaña "Object Alignment" en la paleta de opciones de "Object Selector", como se muestra en la figura 2.

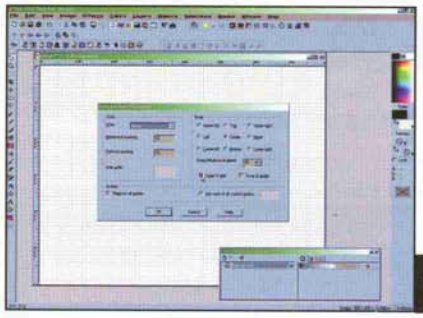
A continuación, vamos a cambiar el grosor de los bordes pulsando en el botón "Properties" de la paleta de opciones del "Object Selector" y cambiamos el "Stroke width" al valor 3 (Fig. 2).

Ya tenemos nuestro cuadrado preparado para posteriormente darle textura. Continuaremos con los dos elementos que faltan. Ambos tienen forma de curva, y la mejor forma de realizarla es a partir de círculos. Con la herramienta "Preset Hapes" anterior, dibujamos un primer círculo que englobe al cuadrado. Y luego otro mayor que englobe al resto de formas, ambos de color negro también (Fig. 3).

Al igual que hicimos con el cuadrado haremos con los círculos: cambiamos el grosor de los bordes con valor 3 y los centramos en pantalla. La siguiente operación será agrupar ambos

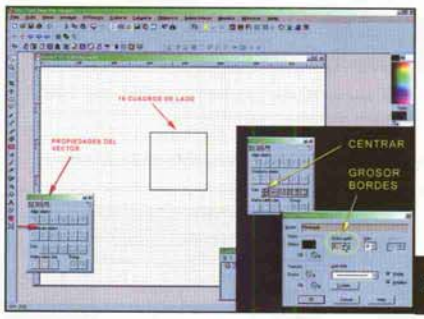
NOTA

Para seleccionar o mover un vector o forma geométrica debemos elegir la herramienta "Object Selector" y pulsar sobre el borde de la figura.



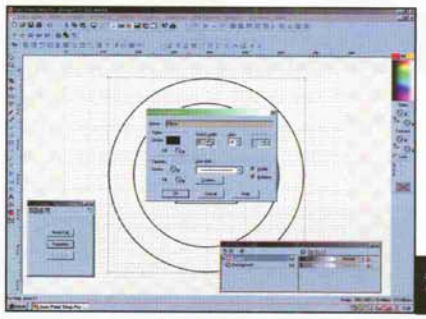
1

Mediante la opción "Snap to Grid" podemos ajustar los movimientos a la cuadrícula.



2

Cada herramienta posee un set de opciones propias. En la figura, el procedimiento para centrar un vector en la pantalla.

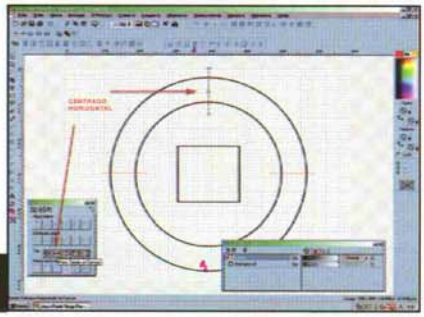


3

En la ventana de propiedades del vector podemos ajustar el grosor del borde del círculo.

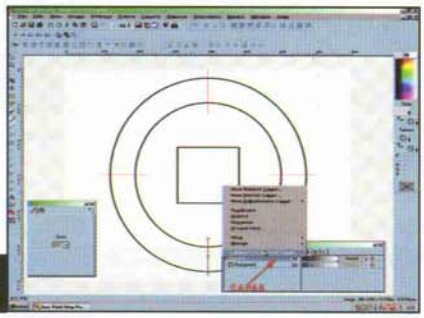
TRUCO

Para lograr un círculo perfecto dejamos pulsada la tecla SHIFT mientras desplazamos el ratón.



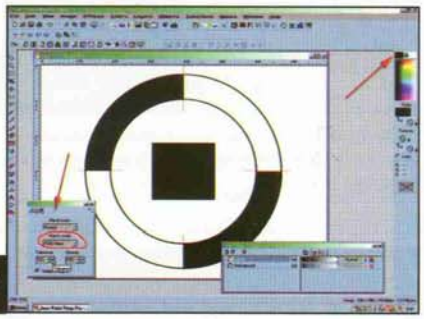
4

Por medio de líneas podemos dividir el círculo en cuatro partes iguales para poder aislarlas.



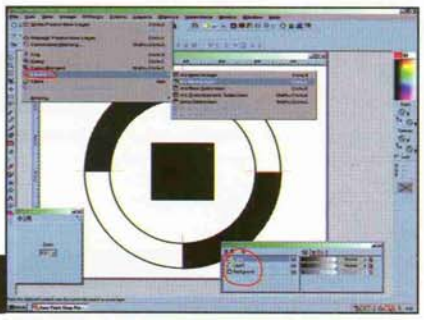
5

Para convertir una capa de vector en una capa de imagen debemos pasterizarla.



6

En las opciones de relleno podemos controlar su cantidad.



7

Mediante la opción "Paste" podemos realizar diferentes tipos de copias de capas.

círculos. Los seleccionamos con la tecla SHIFT pulsada y, a continuación, elegimos la opción "Group" situada en el menú "Object". Esta operación nos ayudará a tratar los dos círculos como si fueran una sola figura.

Las formas que nos interesan son en realidad dos sectores opuestos del anillo formado por los dos círculos. Así que, para aislarlos, dividiremos el anillo en cuatro partes mediante cuatro líneas de color rojo como se muestra en la figura 4.

Podemos observar en la paleta de capas "Layer Palette" cómo nuestras figuras están en una capa sobre el fondo llamada "Layer1". Pues bien, para seguir con nuestro proceso de aislamiento, debemos "Rasterizar" esta capa. Nos situamos sobre ella y pulsamos el botón derecho del ratón. Aparecerá un menú desplegable en el que debemos elegir la opción "Convert to Raster Layer" (Fig. 5).

A continuación, vamos a rellenar de color negro el sector superior izquierdo y el inferior derecho, que son los que nos interesan. Ocultamos la cuadrícula (CTRL+ALT+G) y elegimos la herramienta de relleno. Pero antes realizar esta operación con los sectores y el cuadrado debemos cambiar algunas opciones para obtener resultados óptimos. En el panel de opciones de esta herramienta elegimos un modo de mezcla "Blend Mode" normal, una mezcla de color inicial "Match Mode" a partir de valores RGB "RGB Value", una opacidad del 100 % y para finalizar la tolerancia a 125. En la paleta de colores elegimos el estilo pincel "Styles" y el color negro de tinta y por último rellenamos (Fig. 6).

El siguiente paso será seleccionar las partes rellenas de negro y crear una nueva capa con cada una de ellas (una para los sectores y otra para el cuadrado) para tener definitivamente las partes del logotipo aisladas.

Elegimos la herramienta de selección y le cambiamos las opciones de "Match Mode" a "RGB Value", "Tolerance" a 10 y el tamaño de la pluma de selección "Feather" a 0. Luego seleccionamos pulsando sobre los sectores rellenos de negro con la tecla SHIFT pulsada para sumar selección. (CTRL para restar selección). Nos falta crear una nueva capa con los elementos seleccionamos, para ello copiamos con CTRL + C y pegamos como una nueva capa con CTRL + L, o en "Edit / Paste / As New Layer". Hacemos lo mismo con el cuadrado: lo seleccionamos y creamos otra capa. Podemos observar que en la paleta de capas tenemos tres *Layers*: "Layer1" con todas las figuras rellenas, "Layer2" con los sectores y "Layer3" con el cuadrado (Fig. 7).

La primera capa ya no nos interesa, por lo que pasamos a borrarla desplegando el menú de capas con el botón derecho del ratón y eligiendo "Delete".

Ya tenemos las partes de nuestro logotipo preparadas para aplicarles los efectos necesarios y darles el aspecto final deseado.

NOTA

"Rasterizar" una capa significa convertir su contenido en una imagen editable. Es decir, la capa que hemos creado formada por tres objetos vectoriales, un cuadrado y dos círculos, se convertirá en una capa formada por una imagen de un cuadrado y dos círculos.

DISEÑO DEL ASPECTO DEFINITIVO

Para obtener el aspecto deseado debemos aplicar color, textu-

TRUCO

Para cambiar el nombre de la capa, pulsamos sobre ella y aparecerá su ventana de propiedades.

ras y efectos gráficos a cada parte del logotipo.

En primer lugar aplicaremos un tono azul a los sectores con la herramienta de relleno y un tono rojo al cuadrado. El siguiente paso será la aplicación de texturas. Empezaremos con el cuadrado.

Partiremos de una textura de cuero a la que le aplicaremos diferentes efectos gráficos para obtener un resultado personal (Fig. 8).

Nos situamos en la paleta de colores. Elegimos la opción "pattern" de la caja "Styles" pulsando sobre la flecha negra de "Foreground Color". A continuación, pulsamos en el interior de la caja de relleno. Paint Shop Pro cargará la lista de texturas; entonces elegimos "leather" y rellenamos el cuadrado (Fig. 9).

Rellenaremos ahora los dos sectores del círculo con un degradado. Seleccionamos la capa de uno de los sectores. En "Styles" elegimos ahora el icono "gradient" pulsando sobre la flecha negra. De igual forma, pulsamos en el interior de la caja de relleno y aparecerá una lista con diferentes tipos de gradientes de color. Elegimos el "Blue Metalic" y rellenamos la figura (logo 23). Una vez aplicado el gradiente, le sumaremos un relleno de textura plástica. Situado debajo de "Styles" nos encontramos con la sección "Textures", pulsando sobre la flecha negra elegiremos el icono "Texture", aparecerá la lista de texturas en donde seleccionaremos "Plastic". Rellenamos de nuevo la figura y obtendremos una mezcla de gradiente de color y textura plástica. (Fig. 10).

Realizamos los mismos rellenos para el otro sector del círculo. Y por último, nos queda dar la forma definitiva a estas figuras creándoles una especie de bisel o chaflán en los bordes. Para ello, vamos a aplicarles un efecto "Inner Bevel". En el menú "Effect" elegimos la opción "3D Effects/ Inner Bevel". Aparecerá una ventana con todas las opciones para este efecto. Antes de nada, dejamos pulsado el icono del ojo para mostrar un previo del efecto en la imagen. Jugando con los parámetros se puede modificar el estilo del bisel a nuestro gusto. Cada efecto gráfico en Paint Shop Pro tiene sus propios parámetros de configuración. Las opciones para crear biseles son muy amplias, desde el manejo de la luz a través del color hasta modificar los parámetros generales de la imagen resultante y la forma del bisel. También nos permiten grabar distintas configuraciones para crear nuestra propia librería. Para nuestro logo hemos preparado la configuración que se muestra en la figura 11.

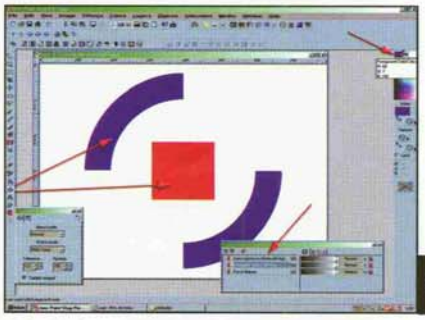
Para terminar, nos queda aplicar el mismo efecto pero con algunas variaciones al cuadrado del centro. Nos falta un efecto más y es modificar la saturación del rojo del cuadrado. Para ello, elegimos "Hue/Saturation/ Lightness" en el menú "Colors/Adjust" (SHIFT + H). Cambiamos la saturación (saturation) a -10 y el brillo (Lightness) a -20. Terminamos rellenando la capa del fondo de color negro.

LA PANTALLA DE PRESENTACIÓN

Ya hemos acabado el logo de "Zone of Fighters". Para completar la pantalla de presentación debemos sumar al logo el título del juego. Guardamos nuestro logo en "File/Save as" con la opción "Windows or OS/2 Bitmap (*.bmp)" con el nombre "logotipo".

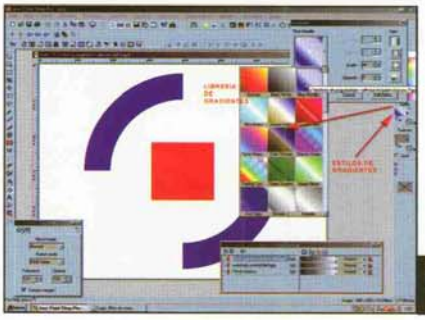
Cerramos la imagen. Cuando se nos pregunte si queremos guardar los cambios pulsamos "Sí" y "Guardar".Tendremos de esta forma una copia del logo pero en formato de capas del Paint Shop Pro para futuras modificaciones.

Cargamos la imagen del logo que guardamos en .bmp. Luego pulsamos en "New" para crear otra imagen nueva con fondo negro y con el mismo tamaño (800x600x72). Vamos a pasar el



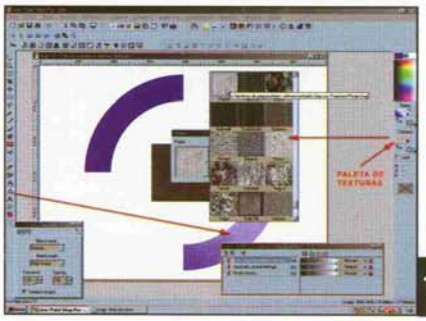
8

Situación de los elementos necesarios para la elección de color y capa para un rellenado.



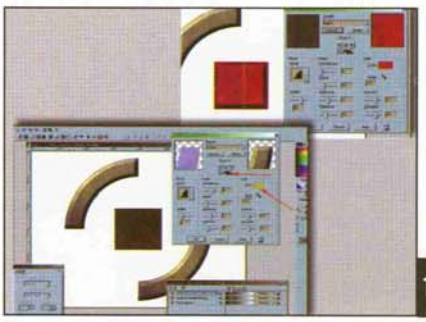
9

Paint Shop Pro posee una librería de gradientes muy extensa, la cual puede ser ampliada por el usuario.



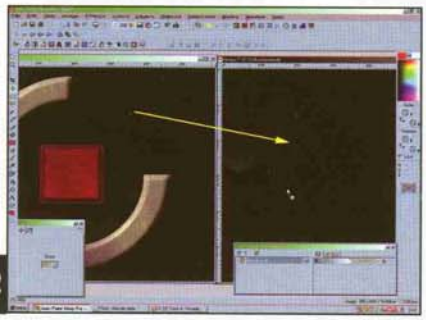
10

Además de gradientes, podemos disponer de una amplia variedad de texturas de relleno.



11

El efecto "Inner Bevel" posee multitud de opciones de configuración para obtener el resultado que buscamos.



12

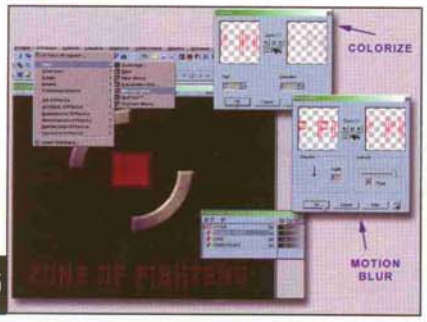
Paint Shop Pro permite pasar una capa de una imagen a otra nueva sólo arrastrándola con el ratón.

13

En la ventana "Text Entry" podemos escribir el texto y además configurar su aspecto y el tipo de letra.

14

Procedimiento que debemos seguir para realizar un buen efecto de desenfoque en el título.



15

Una vez preparadas las capas del título, elegimos el efecto "Motion Blur" para realizar el desenfoque.

logo de la imagen cargada a la nueva imagen vacía creada. Para ello, nos situamos sobre la paleta de capas, colocamos el cursor sobre la capa "fondo" (Background) y desplazamos el ratón, sin dejar de pulsar el botón izquierdo, hacia la ventana de la imagen vacía creada. Observamos cómo en esta nueva imagen aparece una copia del logo en una nueva capa (Fig. 12).

Ya podemos cerrar la imagen del logo, para trabajar con la nueva imagen. Renombramos la capa del fondo como "FONDO NEGRO" y la capa del logo como "LOGO". Debemos ahora reducir el tamaño del logo para dejar espacio en la parte inferior para el título. Para realizar esto, seleccionamos la capa "LOGO" y pulsamos el icono "Deformation". Pulsamos en una de las esquinas y desplazamos para reducir el logo de tamaño. Para escribir nuestro título pulsamos en el icono de texto A. Pulsamos en el lugar de la imagen que queremos escribir y se abrirá entonces la ventana de diálogo "Text Entry". Seleccionamos el tipo de letra *dungeon* y escribimos "Zone of Fighters" de color rojo ("Stroke") como se muestra en la figura 13.

A continuación, cambiamos el grosor de los bordes con valor 2 como hicimos con las figuras del logo. ("Vector Properties").

Vamos a completar el título aplicando un efecto de desenfoque a las letras. Duplicamos la capa del título pulsando con el botón derecho sobre ella y eligiendo la opción "Duplicate". Automáticamente, se crea una nueva capa que debemos rasterizar y colocar debajo de la original, como se muestra en la figura 14.

Esta capa rasterizada (que se ha convertido en una imagen del título) la utilizaremos para aplicar un efecto de desenfoque de movimiento que, al estar debajo de la capa con el título original, quedará por detrás de éste. Antes de seguir, al texto original le aplicamos dos puntos más de grosor con las opciones de vector. Nos situamos de nuevo en la copia del título y aplicamos más luminosidad al color en "Colors/ Adjust/ Hue-Saturation-Lightness" (SHIFT+H) con los valores "Saturation" 25 y "Lightness" 70. Antes del "Motion Blur" vamos a utilizar la función "Colorize" situada en el menú "Colors" para reducir el tono de rojo a prácticamente gris. Para ello vamos a dejar las opciones por defecto de esta función; es decir, es suficiente con una saturación de color a 0.

Ya estamos preparados para aplicar el desenfoque. Elegimos en el menú "Effects" la opción "Motion Blur" del efecto "Blur". Elegimos un ángulo con valor 0 y la intensidad del desenfoque al máximo, es decir 40 píxels. Y para terminar, le volvemos a aplicar el mismo efecto dos veces más (Fig. 15).

NOTA

Si no posees en tu sistema el tipo de letra "dungeon" la puedes encontrar en el CD, apartado "extras". Para instalarla sólo tienes que copiar el fichero a la carpeta "Fonts" situada en el directorio "Windows" de tu sistema operativo Windows. Desde el explorador de Windows, elegir la opción "Actualizar" del menú "Ver" para actualizar la fuente.

En el próximo número...

... realizaremos el tipo de letra que vamos a utilizar en "Zone of Fighters" con la aplicación Font Creator, modificando la fuente "dungeon". También hablaremos un poco de los formatos 3D más comunes utilizados en la realización de juegos y en particular los manejados por "Zone of Fighters".

Cómo desarrollar los efectos especiales de sonido (I)

En la anterior entrega empezamos a conocer la manera de reproducir y editar un sonido en Goldwave. Estudiaremos, a continuación, cómo grabar en nuestro ordenador audio desde el exterior a través de la tarjeta de sonido. Explicaremos la forma de organizar el contenido de la grabación por medio de marcas de posición.

GRABACIÓN DE UN SONIDO

La obtención de audio de una fuente externa es la mejor forma de empezar a crear un sonido propio. Aplicando los efectos pertinentes, podemos convertir, por ejemplo, una palmada en una terrible explosión. Todo depende de nuestra imaginación, de lo demás se encargará el editor de audio.

Antes de comenzar a grabar, debemos conocer todas las opciones que nos ofrece el programa para realizar correctamente esta acción.

PREPARAR EL DISPOSITIVO DE AUDIO

Partimos de que tenemos instalada una tarjeta de sonido en el ordenador y que hemos conectado un micrófono a la entrada "Mic" de la tarjeta. Creamos una plantilla de sonido nueva para que aparezcan las opciones.

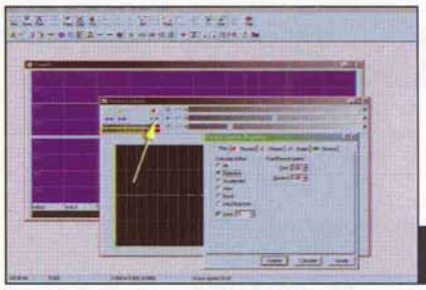
En primer lugar, nos situamos en la pestaña de dispositivos "device" del panel de propiedades de control de dispositivos "Device Controls Properties". Este panel aparece pulsando en el icono de la ventana "Device Controls". En "Device" configuramos las opciones para las acciones de reproducir y grabar. En el apartado "Playback device" se muestra una lista de los controladores de los dispositivos de audio disponibles para reproducir y en "Record device" los que podemos seleccionar para grabar (Fig. 1).

Debajo de cada una de estas listas se encuentran las posibilidades de cada uno de los controladores o drivers. Horizontalmente encontramos la frecuencia de muestreo posible y en vertical la calidad. Los grupos de dos cuadraditos que están debajo de cada número de frecuencia representan con color verde si el controlador elegido soporta las cualidades que acompaña. Si ambos cuadrados de cada grupo están en verde, el controlador soporta sonido estéreo y si sólo lo está uno de ellos, sonido en mono. Con estos indicadores controlaremos qué atributos sonoros podemos obtener del controlador elegido (Fig. 2).

Encontramos también un apartado para controlar el tamaño del buffer, que no es más que una zona de memoria reservada para reproducir o grabar. Este tamaño viene determinado en segundos y va desde 0 hasta 5. Por defecto, se asigna 1 segundo a cada buffer y generalmente no conviene subir de ahí.

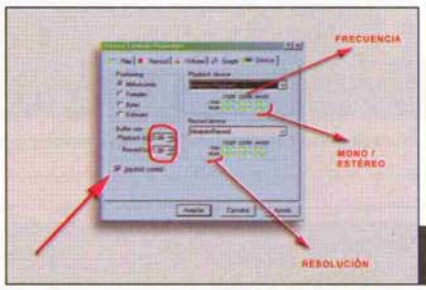
PREPARAR LA GRABACIÓN

Vayamos ahora a la pestaña "Record". Encontramos numerosas opciones que nos ayudarán a preparar la grabación adecuadamente. En el apartado de opciones de grabación "Recording Options" encontramos opciones muy interesantes y útiles. La casilla "Monitor" activa o desactiva el gráfico de medida de volumen. Si activamos "Loop", cuando se llegue al final del tiempo establecido para grabar, continuará de nuevo desde el inicio añadiéndose a lo ya grabado, y así sucesi-



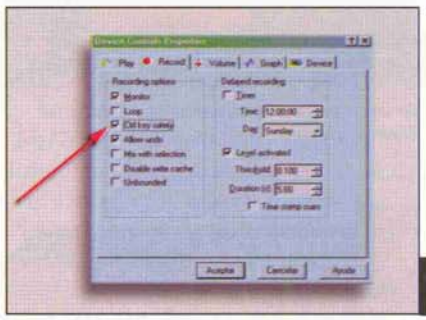
1

En "Device Controls" encontramos el botón de opciones de configuración.



2

Opciones de configuración del dispositivo de audio.

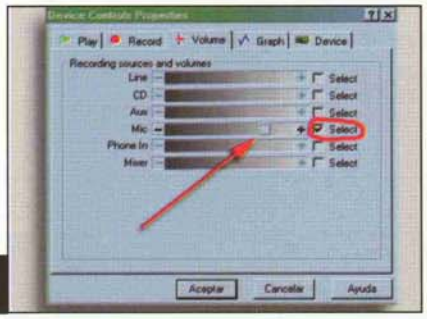


3

Ventana de configuración de las opciones de grabación.

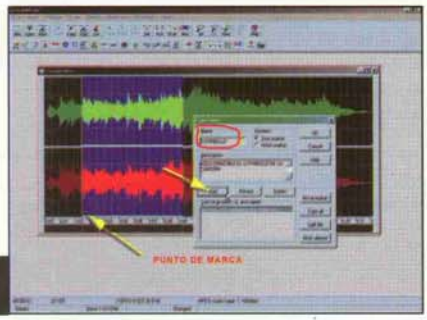
NOTA

Para obtener mayor calidad hay que evitar elegir el mapeador de sonidos de Microsoft (Microsoft Sound Mapper). Sólo seria necesario seleccionarlo si se utiliza una tarjeta de sonido de 8 bits.



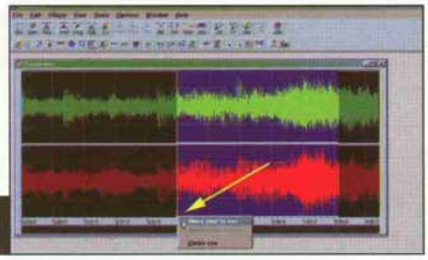
4

La regulación adecuada del volumen del micro es importante para evitar saturaciones en la grabación.



5

Procedimientos para generar puntos de marca.



6

De una manera sencilla podemos desplazar las líneas de selección hacia los puntos de marca.

TRUCO

Activando la casilla "Joystick control" situada en la pestaña "Device" podemos controlar la reproducción y grabación con el joystick o gamepad que haya instalado en el ordenador. Para reproducir o parar se utiliza el botón primario y para grabar CTRL + botón secundario. Para controlar la velocidad y dirección de la reproducción se utiliza el control de dirección hacia delante o hacia atrás

vamente hasta que se pulse "Stop". "Ctrl key safety" es para seguridad y activa o desactiva el hecho de tener que dejar pulsado la tecla CTRL para grabar. "Mix with selection" activada, permite realizar capas de sonidos; es decir, si grabamos sobre un sonido, no lo borra sino lo mezcla con la nueva grabación. Activando "Disable write cache" permitimos que el nuevo sonido se vaya escribiendo en el disco duro mientras grabamos. Es muy útil para realizar una grabación de gran duración, siempre que dispongamos de un disco duro lo bastante rápido (Fig. 3).

En la pestaña "Volume" podemos seleccionar el volumen de grabación de las diferentes entradas de audio permitidas. Seleccionamos la casilla "select" de "Mic" y regulamos el volumen más o menos al 80%. Lo ideal es hacer pruebas con el micrófono y controlar que el vúmetro no llegue al rojo (Fig. 4).

¡SILENCIO POR FAVOR... GRABANDO!

Ya conocemos las opciones más importantes para poder empezar a grabar correctamente. A continuación, necesitamos crear una plantilla vacía para insertar nuestra grabación. Pulsamos en "New" (CTRL + N) para abrir la ventana "New sound". Vamos a grabar la palabra "MI PRIMERA GRABACIÓN" en estéreo con calidad de CD, es decir 44100 Hz de frecuencia y 16 bits de resolución, así que pulsamos sobre el botón "CD". Seguidamente, definiremos la longitud de la plantilla a 5 segundos en "Lenght (MM:SS:TTT)". Pulsa el botón rojo y... ¡a grabar!

Ahora sólo queda probar a cambiar las opciones de grabación para comprender mejor su utilidad.

DEFINICIÓN

▸ GRABAR CON CACHÉ

Significa que el sonido se almacena en memoria antes de pasarlo al disco duro.

ORGANIZAR UN SONIDO

Vamos a conocer una utilidad que tiene Goldwave para señalar aquellas partes del sonido o música que interesan y que, en cierta manera, ayudará a organizar mejor el trabajo, sobre todo para la edición de partes. Nos referimos a los puntos de marca "Cue Point". Para colocar una de estas indicaciones, debemos situar primero la marca de principio de selección en el lugar donde queremos colocar el punto.
Posteriormente la opción "Cue Point" del menú "Tools".
Pinchamos la casilla "Start marker" y en "Name" escribimos el nombre del punto o una pequeña descripción. Y para terminar, pulsamos en el botón "Add" para sumar el punto (Fig. 5).

Observamos cómo la indicación aparece en la posición de la marca de principio de selección en la parte inferior con forma de banderita azul. Desplazando el ratón con el botón izquierdo pulsado, mover la banderita de sitio. Para borrar el punto, podemos hacerlo desde la ventana "Cue Points" o bien pulsando con el botón derecho del ratón sobre la banderita y eligiendo la opción "Delete cue". En este mismo menú emergente, encontramos dos opciones más: "Move start to cue" y "Move finish to cue", que nos permite situar las marcas de principio y final de selección en la posición del punto elegido (Fig. 6).

Podemos crear cuantos puntos indicadores queramos. Son útiles, por ejemplo, para señalar el comienzo y final de cierta parte de una canción y, posteriormente, situar las marcas de selección en ellos y copiarla.

En el próximo número...

... empezaremos a realizar nuestros propios sonidos. Practicaremos la aplicación de filtros y efectos especiales y conoceremos el evaluador de expresiones.

Tipos de datos, funciones gráficas y sistema

ESTRUCTURAS DE DATOS

Ya conocemos cómo controlar el flujo de nuestro programa y cómo trabajar con matrices. Pero hay un sistema muy eficaz para manejar datos de un mismo tipo: nos referimos a las estructuras de datos o *Type*. Supongamos que queremos dibujar un cielo estrellado con 200 estrellas. Cada estrella está situada en una posición X (horizontal) e Y (vertical) distinta, pero todas tendrán los mismos tipos de variables. Podemos utilizar matrices para definirlas:

```
Dim estrella_X(199),estrella_Y
(199),estrella_grafico(199)
```

Este sistema no es muy cómodo que digamos, así que utilizaremos una estructura. Para definir estructuras se utiliza en Blitz3D las instrucciones Type, Field y End Type. Vamos a crear un tipo *estrella* que servirá para describir cada estrella:

```
Type tipo_estrella
  Field X,Y
  Field grafico
End Type
```

Con "Field" creamos los campos para almacenar las variables que usaremos en la estructura y con "End Type" la cerramos.

Hay que tener en cuenta que esta estructura sólo describe a una estrella. Para crear nuevas estrellas debemos utilizar la instrucción "New", de la siguiente forma:

```
estrella.Tipo_estrella=
New Tipo.estrella
```

En la nueva variable "estrella", almacenamos a la nueva estrella ("Tipo_estrella").

Para acceder a los campos de la nueva estrella utilizaremos el símbolo " \ ", por ejemplo, asignemos la posición 100 horizontal a la nueva estrella: estrella\X=100.

Con los campos podemos realizar las mismas operaciones, como si de una variable cualquiera se tratara, por ejemplo: estrella \ X = estrella \ X + 10 . Incrementamos en 10 la coordenada X de la estrella. Ya tenemos creada una nueva estrella, pero ¿cómo creamos las otras 199? Pues bien, todas las demás pueden ser también del *tipo_estrella*; esto es, crearemos una colección de 200 estrellas con sus campos comunes, posición (X,Y) y gráfico (grafico). A continuación, crearemos 200 estrellas y les asignaremos una posición aleatoria en la pantalla; por el momento, su aspecto gráfico será el de un asterisco "*":

```
For n = 1 to 200
    estrella.tipo_estrella  =
  New tipo_estrella
    estrella\X = Rnd (0,640)
    estrella\Y = Rnd (0,480)
    estrella\grafico$ = "*"
Next
```

Si queremos mover todas las estrellas de sitio, Blitz3D dispone de un bucle especial que recorre la lista de tipos, accediendo a cada elemento creado. Cambiemos las posiciones de todas las estrellas:

```
For estrella.tipo_estrella  =
  Each tipo_estrella
      estrella\X = Rnd (0,640)
      estrella\Y = Rnd (0,480)
Next
```

Para borrar una estrella de la lista utilizaremos la instrucción "Delete": Delete estrella

Y para borrar todas las estrellas "Delete Each": Delete Each estrella

Para navegar por la lista de tipos, disponemos de cuatro instrucciones: First, Last, After y Before.

El operador "First" retornará el objeto que está en el principio de la lista y "Last" retornará el último, por ejemplo:

```
estrella.tipo_estrella =
  First tipo_estrella
estrella.tipo_estrella =
  Last tipo_estrella
```

```
Graphics 800,600,16,2
Type tipo_estrella
 Field x,y  ; Coordenadas del grafico
 Field grafico$ ; grafico de la estrella
End Type
For n=1 To 100 ; creamos 30 estrellas
   estrella.tipo_estrella=New tipo_estrella
   estrella\x=Rnd(0,800)
   estrella\y=Rnd(0,600)
   estrella\grafico$="*"
   Text estrella\x,estrella\y,estrella\grafico
Next
Repeat
  If KeyDown(57)
    Delay 200 :  Cls
    For estrella.tipo_estrella=Each tipo_estrella
     estrella\x=Rnd(0,640)
     estrella\y=Rnd(0,480)
     Text estrella\x,estrella\y,estrella\grafico
    Next
  EndIf
 Until KeyDown(1)  ; Salimos si pulsamos Escape
```

1

Un simple ejemplo en el que se muestra la creación de tipos.

```
Graphics 640,480,16,2
Type tipo_letra
 Field x,y  ; Coordenadas del grafico
 Field grafico$ ; grafico de la letra
End Type
Dim letra.tipo_letra(4)  ; 5 letras diferentes con el mismo tipo de datos
For n=0 To 4
   letra(n)=New tipo_letra
   letra(n)\x=Rnd(0,640)   ; Posicionamos cada una de las letras
   letra(n)\y=Rnd(0,480)
Next
; Asignamos una letra a cada letra
letra(0)\grafico$="A"
letra(1)\grafico$="B"
letra(2)\grafico$="C"
letra(3)\grafico$="D"
letra(4)\grafico$="E"
For n=0 To 4
   Text letra(n)\x,letra(n)\y,letra(n)\grafico
Next
WaitKey()
```

2

Un ejemplo de dimensionado de estructuras.

Si la lista de tipos estuviera vacía, estos operandos devolverían un valor "Null" o cero.

Podemos también acceder a un objeto posterior o anterior en una lista con los operandos "After" y "Before":

```
Estrella.tipo_estrella =
 First estrella ➡ Primera
 estrella de la lista
estrella  = After  estrella ➡
 Segunda estrella de la lista
estrella  = Before  estrella ➡
  Primera estrella de la lista
  de nuevo
```

Si queremos insertar un objeto en la lista debemos utilizar la instrucción "Insert". Vamos a colocar una estrella al principio de la lista de tipos:

```
Insert estrella  Before First
tipo_estrella
```

Podemos también crear un array o matriz de tipos. Imaginemos que queremos definir 20 coches de carreras diferentes. Cada uno utiliza una misma estructura de datos:

```
Type tipo_coche
  Field X,Y
  Field color_coche
  Field grafico
End Type
```

Si los creamos con un bucle:

```
For n = 1 to 20
   coche.tipo_coche = New tipo_coche
   coche\X = Rnd (0,640)
   coche\Y = Rnd (0,480)
   coche\grafico = gráfico del coche
Next
```

No podemos acceder a uno de ellos en concreto porque todos se llaman "coche". Tendríamos que asignarles un nombre diferente para cada uno:

```
coche1.tipo_coche = New tipo_coche
coche2.tipo_coche = New tipo_coche
coche3.tipo_coche = New tipo_coche
.
.
.
coche20.tipo_coche = New tipo_coche
```

¡Ufff , mucho trabajo! Utilizaremos entonces un array:

```
Dim coche.tipo_coche(19)
For n = 0 to 19
   coche(n)  =  New tipo_coche
   coche(n) \ X = Rnd (0,640)
   coche(n) \ Y = Rnd (0,480)
   coche (n)\ grafico = gráfico
   del coche
Next
```

Por ejemplo, para incrementar en 20 la coordenada X del quinto coche sería:

```
coche(4) \ X = coche(4) \ X + 20
```

Hasta aquí el tema de las estructuras de datos. Al principio puede parecer complicado, pero es la manera más cómoda de trabajar con Blitz3D. Debes practicar mucho y hacer miles de pruebas. De todas formas se pueden sustituir las estructuras por matrices y obtener los mismos resultados. Los arrays son generalmente utilizados por programadores de BASIC, mientras que las estructuras son propias del lenguaje C. Por ejemplo, en la programación de "Zone of Fighters" se ha utilizado ambos métodos: arrays, tipos y arrays de tipos.

En el CD-ROM (apartado "extras") podrás encontrar dos pequeños ejemplos que ilustran lo visto hasta ahora: "ejemplo5_1.bb" y "ejemplo5_2.bb".

FUNCIONES GRÁFICAS

Antes de ejecutar cualquier programa en Blitz3D, debemos conocer algunas de las funciones gráficas que hacen posible la representación en pantalla de cualquier texto o gráfico 3D, así como las técnicas que Blitz3D utiliza para lograr suavidad, velocidad y calidad en la imagen.

CONFIGURACIÓN DEL MODO GRÁFICO

Antes de ejecutar cualquier instrucción gráfica hay que configurar la resolución, profundidad de color y modo de la pantalla 3D. Para ello, disponemos de la instrucción "Graphics3D".

Sintaxis:

```
Graphics3D Resolución
Horizontal en píxels,
Resolución Vertical,
Profundidad de color
(16, 24 o 32 bits), modo de
representación
```

Ejemplo: Graphics3D 640,480,16,1:

El modo de representación es opcional, y sus valores son:

- **0:** Modo ventana.
- **1:** Siempre en pantalla completa.
- **2:** Siempre en modo ventana.
- **3:** Siempre en modo ventana escalable.

El *modo ventana* en la opción cero sólo funciona con la opción "debug" del editor activada.

Ocurre que no todos los equipos poseen la misma tarjeta gráfica, así que es necesario saber qué configuración podemos implantar en un sistema. Primero vamos a averiguar si la tarjeta gráfica es capaz de soportar opciones 3D. Debemos obtener una lista de todas las tarjetas gráficas instaladas en el equipo con la

función "CountGfxDrivers()" y, a continuación, preguntar sus capacidades 3D con la función "GfxDriver3D()". Para obtener el nombre de la tarjeta gráfica utilizaremos la función "GfxDriverName()":

```
For n= 1 to CountGfxDrivers()
  If GfxDriver3D(n)=True
      Print "La tarjeta " +
GfxDriverName(n) + "si tiene
capacidad 3D"
    Else
      Print "La tarjeta " +
GfxDriverName(n) + " no tiene
capacidad 3D"
    Endif
Next
```

La misma operación podemos realizarla para averiguar los modos gráficos que soporta la tarjeta y en cuáles funcionan las capacidades 3D. Para ello utilizaremos las funciones "CountGfxModes() " y "GfxMode3D ()". Además, contamos con otras tres funciones que nos ayudarán a averiguar con exactitud la resolución horizontal, vertical y la profundidad de color de cada modo gráfico posible:

```
" GfxModeWidth() "(horizontal),
" GfxModeHeight() " (vertical),
"GfxModeDepth() " (color)
```

Con todas estas funciones podemos realizar un pequeño programa para saber los modos gráficos soportados y todas sus características:

```
For n= 1 to CountGfxModes()
  If  GfxMode3D(n)=True
      Print "El modo " + n  +
" si tiene capacidad 3D"
      Print "Resolución
horizontal del modo: " + n +
" =  " +  GfxModeWidth(n)
      Print "Resolución
vertical del modo ... : " + n
+ " =  " + GfxModeHeight(n)
      Print "Profundidad de
color del modo .:  " + n
+ " = " + GfxModeDepth(n)
    Else
      Print "El modo " + n
+ " no tiene capacidad 3D"
    Endif
Next
```

Una función "Windowed3D()" nos avisa si la tarjeta gráfica soporta o no el modo de ventana:

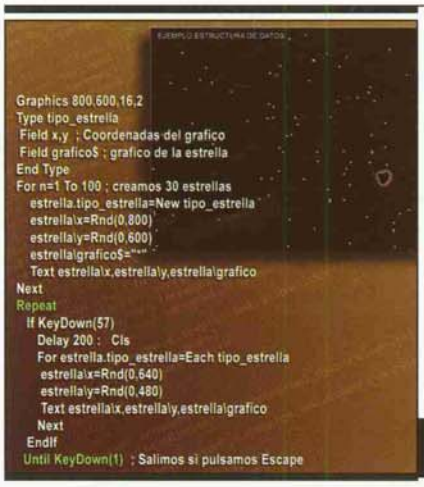
```
If Windowed3D() = True then
Print "Si soporta el modo
ventana"
```

Disponemos también de una última función que nos comunica directamente si un modo gráfico es soportado por la tarjeta:

```
GfxMode3DExist (resolución
Horizontal, Resolución Vertical,
Profundidad de Color)
If GfxMode3DExist(2048,1040,32)=
True Then Print "Si soporta
esta resolución"
```

BÚFER Y DOBLE BÚFER

La magia escondida detrás del manejo de los gráficos de un videojuego la encontramos en el búfer de vídeo. Éste no es más que una zona de memoria que contiene gráficos. En Blitz3D disponemos de dos de ellos: uno visible llamado "FrontBuffer" y otro invisible llamado "BackBuffer". Si dibujamos directamente en el "Frontbuffer" (búfer frontal) lo estaremos viendo en la pantalla al mismo tiempo, así que todos los cambios que realicemos los veremos inmediatamente. Además, estos cambios permanecerán en la pantalla todo el tiempo. Por ejemplo, si nuestra nave avanza por un fondo estrellado, observaremos cómo borra todo a su paso. Utilizando esta técnica, hay que reponer el trozo de fondo por donde pasa la nave cada vez que se mueve. Para evitar este suplicio, existe la técnica del doble búfer, que consiste en utilizar dos zonas de memoria para



Quizás el ejemplo 3D más simple.

dibujar, la visible y la invisible, es decir, en Blitz3D, dibujaremos primero los cambios del gráfico en el "BackBuffer" para posteriormente pasarlo al "FrontBuffer" (la pantalla). Con este sistema, el jugador ve moverse en pantalla un gran número de gráficos de forma suave y sin alterar el fondo.

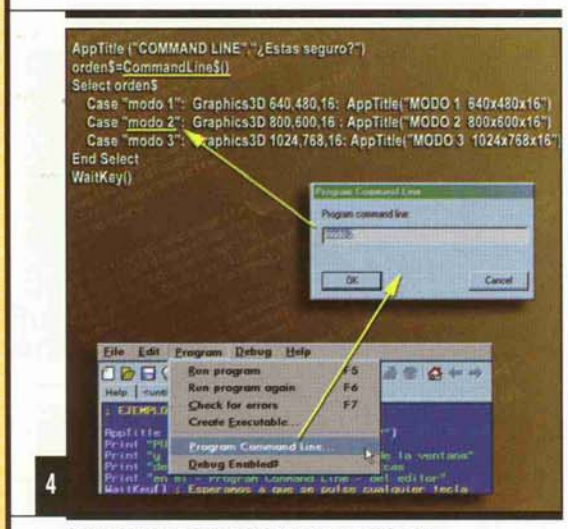
Por defecto, está activado el "FrontBuffer", por ello, dibujaremos en tiempo real sobre la pantalla. Pero si queremos activar el doble búfer debemos utilizar la instrucción "SetBuffer":

```
SetBuffer BackBuffer()
```

Activado el "BackBuffer" como zona por defecto, ningún cambio gráfico que realicemos se verá en pantalla. Para poder verlos hay que pasarlos al búfer frontal con la instrucción "Flip":

```
Repeat
    Cls
    Movemos el gráfico
    Flip
    Dibujamos el gráfico
Forever
```

La instrucción "Cls" permite borrar la pantalla (búfer) y es necesario antes de permutar el búfer para que no se vean acumulados los cambios.



4

Cómo utilizar la función "CommandLine$".

DIBUJAR MI MUNDO 3D

Al inicializar los gráficos en 3D con "Graphics3D" estamos creando un mundo en tres dimensiones donde el ordenador dibuja (renderiza), frame a frame, todo en tiempo real. Hasta que este procedimiento de renderizado no se realice, no veremos ninguno de nuestros gráficos 3D en pantalla. Así que tenemos que decirle al ordenador que dibuje nuestro mundo con la instrucción "RenderWorld". Sin embargo, todos los cambios que hagamos con nuestros objetos 3D (movimientos, detección de colisiones o transformaciones) seguirán sin verse a no ser que actualicemos el mundo 3D con la instrucción "UpdateWorld":

```
Graphics3D 640,680,16
SetBuffer BackBuffer()
Creamos una camara
Creamos un cubo
Repeat
    Movemos el cubo
    UpdateWorld
    RenderWorld
    Flip
Forever
```

El "programa3.bb" del CD es un pequeño ejemplo del funcionamiento de todo este proceso.

Antes de terminar este apartado, es interesante saber que disponemos de una función para poder realizar una captura de la pantalla en un archivo de mapa de bits o bmp. Se trata de "SaveBuffer".

Esta función salva el contenido del búfer especificado:

```
SaveBuffer (FrontBuffer(),
"captura.bmp")
```

FUNCIONES DE SISTEMA Y DEPURADO

Blitz3D nos provee de una función para comunicarnos con nuestro programa desde Windows. Imaginemos que podemos darle a nuestro juego una instrucción antes de ejecutarse, por ejemplo, con qué resolución gráfica debe comenzar. Utilizaremos para ello la función "CommandLine$()". Veamos el código de nuestro ejemplo:

```
orden$ = CommandLine$()
Select orden$
    Case "modo 1": Graphics3D
    640,480,16
    Case "modo 2": Graphics3D
    800,600,16
    Case "modo 3": Graphics3D
    1024,768,16
End Select
Waitkey()
```

Para probar este programita desde el editor, debemos escribir "modo 1", "modo 2" o "modo 3" en la línea de texto disponible en la ventana "Program Command Line" situada en el menú "Program" y luego ejecutar (F5). Observaremos que lo escrito en la línea de comando pasa a la variable "orden$" a través de la función "CommandLine$()" y según sea, activamos uno u otro modo gráfico.

Otra función útil es "RuntimeError mensaje$". Mostrará el "mensaje$" de error en una ventana de Windows con título "Runtime error".

Precisamente, para dar título a la ventana de ejecución disponemos de la instrucción "AppTitle". En el siguiente ejemplo damos título a la ventana y preguntamos al cerrarla si queremos salir o no:

```
AppTitle ("MI PROGRAMA",
"¿Estás seguro?")
```

Se pueden ver estos ejemplos en el "ejemplo5_4.bb" del CD-ROM.

Para ocultar o mostrar el puntero del ratón en el modo ventana tenemos las instrucciones

"ShowPointer" (mostrar puntero) y "HidePointer" (Ocultar puntero).

Y para terminar, una función muy interesante que permite crear mensajes de error en tiempo de ejecución. Nos estamos refiriendo a "DebugLog mensaje$". Por ejemplo, en una parte del programa, preguntamos si la tarjeta gráfica soporta una determinada resolución. Si no es así, podemos escribir una nota del error para nuestro control:

```
Graphics3D 2048,1024,64
If GfxMode3DExist(2048,1040,64) =
False
  DebugLog "La tarjeta no ha
  soportado esta resolución"
Endif
```

Para ver el aviso pulsamos F9 desde el editor.

Ya estamos preparados para conocer a fondo todas las posibilidades gráficas que nos ofrece Blitz3D.

En el próximo número...

... comenzaremos a estudiar sus características 2D, empezando por las funciones de texto e imagen.

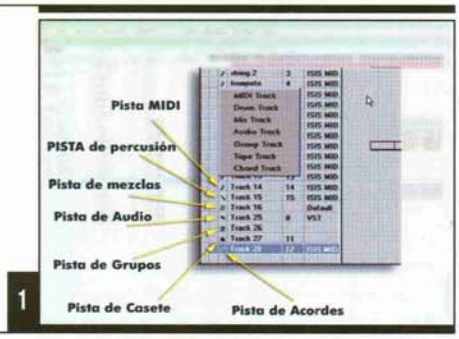
Secuenciadores. Cubase VST (II)

En el primer capítulo de este tutorial de Cubase dimos los primeros pasos para conocer la interfaz de este estupendo secuenciador. En esta segunda entrega aprenderemos a trabajar en la ventana de arreglos y a utilizar las herramientas de edición. Antes de empezar, vamos a abrir Cubase y a cargar la canción del CD-ROM ("tutorial.all").

TRABAJAR CON PISTAS EN LA VENTANA DE ARREGLO

Toda la información grabada en Cubase se almacena en las pistas. Éstas, a su vez, están formadas por partes, las cuales contienen eventos. Entre pistas se puede mover y copiar información. Se pueden crear todas las pistas que se quiera. Trabajar con diferentes pistas nos ayuda a orquestar un tema y tener todas sus partes divididas listas para editar. En la figura 1 se muestran los tipos de pistas que podemos crear.

Para crear una pista nueva basta con pulsar CTRL+T o hacer clic en la parte vacía debajo de la lista de pistas. Para duplicar, mantenemos pulsada la tecla ALT y arrastramos la pista que queremos copiar a la zona vacía de la lista de pistas. Si lo que deseamos es cambiar el orden de una pista, basta con desplazarla con el ratón, manteniendo pulsado el botón izquierdo. Y para borrarla, la seleccionamos y pulsamos la tecla DEL o "Edit / Delete Track".



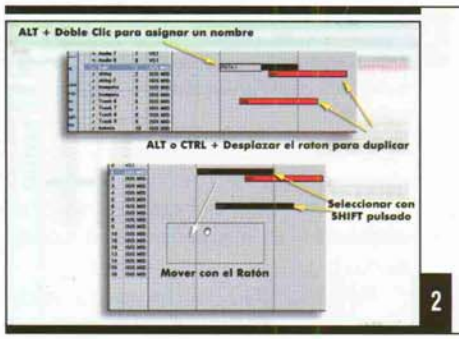
Un esquema de los distintos tipos de pistas que podemos crear en Cubase.

TRABAJAR CON PARTES EN LA VENTANA DE ARREGLO

Una parte se crea automáticamente al grabar algo. Pero, en ocasiones, necesitaremos crear partes vacías para dibujar en ellas nota a nota. Para crear una parte colocamos los localizadores en el comienzo y fin del tamaño de la parte a crear y elegimos la opción "create Parts" del menú "Structure" o simplemente hacemos doble clic entre los localizadores a la altura de la pista. Una nueva parte se creará entre los localizadores. Si queremos renombrarla hacemos doble clic sobre el nombre de la parte o track con al tecla ALT pulsada.

Todas las funciones de edición como mover, duplicar, borrar, son idénticas en todas las ventanas y modos de edición. Así, para mover las partes de lugar basta con desplazarlas con el ratón manteniendo pulsado el botón izquierdo. Para duplicarlas, realizamos la misma operación pero con la tecla ALT pulsada. Si lo que deseamos es borrar, seleccionamos la parte y pulsamos la tecla DEL o SUPR.

Hay varias maneras de seleccionar varias partes a la vez: englobándolas en un rectángulo realizado con el ratón, seleccionar las partes con la tecla SHIFT pulsada o con el cursor derecho después de seleccionar una parte (en una misma pista) (Fig. 2).

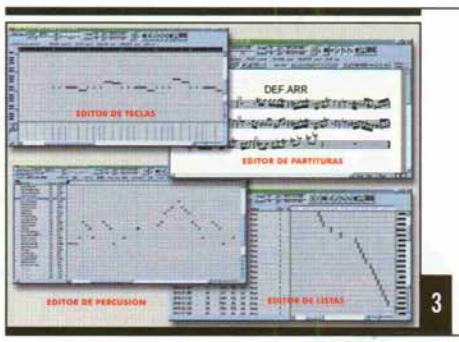


Procedimientos sobre pistas: asignación de nombres, selección y desplazamiento.

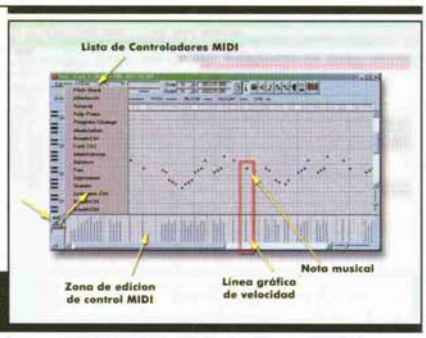
Las funciones de edición descritas anteriormente funcionan igualmente para una parte que para varias seleccionadas.

USO DE HERRAMIENTAS

El uso de herramientas es una manera más cómoda de editar en Cubase. Cada ventana de edición posee sus propias herramientas, las cuales se encuentran agrupadas en grupos de iconos englobados en pequeñas ventanas flotantes. Estas cajas de herramientas aparecen al pulsar el botón derecho del ratón y desaparecen al soltarlo. Para seleccionar una herramienta, recorremos



Los distintos tipos de editores o ventanas de edición en Cubase.



4

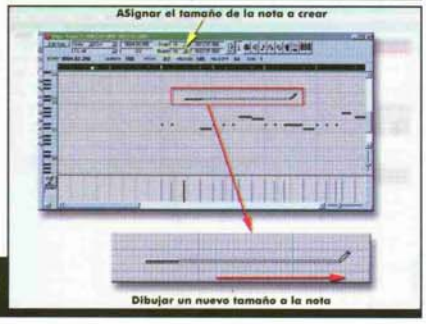
Descripción de las principales partes del editor de teclas.

los iconos con el botón derecho pulsado y lo elegiremos al soltar dicho botón.

Por ejemplo, en la ventana de arreglo, podemos crear una parte simplemente dibujándola con la herramienta, o borrarla con la goma. También podemos dividir una parte haciendo clic con la herramienta de corte. Otra opción interesante es la unión de partes con la herramienta de pegamento. Incluso podemos oír los eventos que deseemos, moviéndonos (con el botón izquierdo pulsado) sobre las partes con la herramienta o mutarla para que no suene con.

USO DE HERRAMIENTAS EN LA VENTANA DE EDICIÓN DE TECLAS

Los eventos de cada parte pueden ser editados a través de las ventanas de edición. Hay varias de estos editores: Key Edit (editor de teclas), Drum Edit (editor de percusión), List Edit (editor de listas) y Score Edit (editor de partituras) (Fig. 3).



5

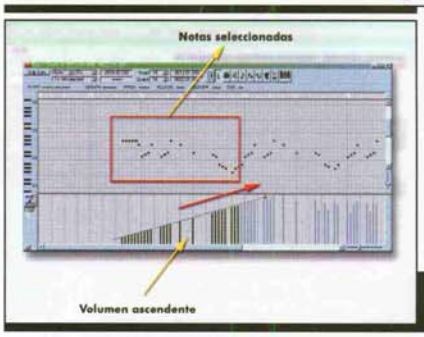
Procedimiento para dibujar una nota con la herramienta del lápiz.

Cada uno de ellos presenta características y herramientas diferentes acorde al tipo de eventos que manejan, pero básicamente, los procedimientos de edición y uso de herramientas en todos ellos es el mismo. Por ejemplo, podemos ver y mover la misma nota en el editor de teclas, en el de listas o en el de partituras. Simplemente cada uno muestra esa nota de una forma diferente. Vamos a practicar con el editor de teclas, que es el más utilizado. Aquí podemos realizar todos los cambios que queramos en cada uno de los eventos MIDI que hayamos grabado (tonos, volúmenes, fuerza de pulsación del teclado, etc). Para entrar en la ventana de edición de teclas basta con hacer doble clic sobre la parte que queremos editar o seleccionar la parte y pulsar CTRL + E. Se puede ver una descripción de las partes de este editor en la figura 4.

Para dibujar o colocar una nota podemos utilizar la herramienta del lápiz (la longitud o duración de la nota la definimos en la casilla "Quant") simplemente dibujamos con él. Para borrar usamos la goma y para mover, copiar o seleccionar usaremos los mismos procedimientos explicados anteriormente para las partes (Fig. 5).

Podemos también cambiar la duración de una nota, con sólo dibujar el nuevo tamaño haciendo clic en el borde derecho de dicha nota.

Una cualidad muy importante de este editor es la posibilidad de modificar no sólo la nota o tono sino cualquier evento MIDI que se haya grabado, como la velocidad de pulsación, el cambio de la rueda de tono o el uso del pedal de *sustain* del teclado, entre otros. Practiquemos, por ejemplo, modificando la velocidad de una nota. Esta cualidad viene implícita con cada nota que se graba y representa la fuerza de pulsación de la tecla (en ocasiones se reconoce por el aumento o disminución del volumen de la nota).



6

Procedimiento para realizar una subida de volumen modificando la velocidad de la nota.

La visualización gráfica de los controladores MIDI la encontramos en la parte inferior de la ventana, dividida de las notas por una línea. Pulsando el icono podemos ver la lista de controladores MIDI que podemos representar (por defecto, aparece el controlador de velocidad).

Para cambiar la velocidad de una nota utilizaremos la herramienta del lápiz subiéndolo o bajándolo sobre la línea gráfica de velocidad. Para cambiar de nota podemos usar las teclas derecha e izquierda. Un uso muy frecuente de esta propiedad es la creación de subidas y bajadas de volumen en determinada porción de notas. Para implementar este procedimiento se puede utilizar la herramienta de línea. Con esta herramienta podemos dibujar una línea ascendente o descendente en la gráfica de velocidad, cambiando los valores, como se muestra en la figura 6.

También es muy útil esta herramienta para igualar la velocidad de un grupo de notas.

Para cambiar los demás controladores MIDI se utilizan las mismas herramientas y se dibujan las modificaciones del mismo modo.

En el próximo número...

... terminaremos estos tutoriales sobre el Cubase aprendiendo a grabar eventos MIDI, a trabajar el editor GM/GS/XG, y sobre todo a cuantizar.

El fin del Spectrum y la llegada del PC

En este número continuamos viendo la evolución del ZX Spectrum, a partir de 1985, que fue cuando se encontraba en pleno apogeo, y contaba con un montón de juegos y no menos seguidores.

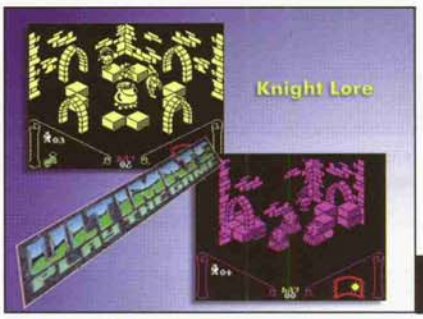
1985

Este fue el año de la consagración de los juegos para Spectrum. Ya se conocía la gestión de *sprites* y la técnica *filmation* de Ultimate. Los juegos aprovechaban al máximo los 48 Kb del ZX- Spectrum. Aparece la continuación de *Knight Lore*, *Alien 8* y cuando parecía que todo estaba inventado publicaron *Night Shade* utilizando la nueva técnica *filmation 2*, mientras que muchas otras desarrolladoras seguían calcando la isometría de Ultimate. Un nuevo género era demandado cada vez más, los juegos de peleas. El beat'em up que introdujo *Kung Fu* fue seguido masivamente. Aparecieron verdaderas maravillas, con gráficos de movimientos suaves y variados, como: *The Way of the Explondiya Wallyg Fist* de Malbourne House, *International Karate* (*System 3*) o *Yie Ar Kung Fu* (Imagine). Se puso de moda el género deportivo tras las olimpiadas y reinaron títulos como *HyperSporst* (Imagine), *Daley Thompson's Supertest* (Ocean) o *Match Day* de fútbol (Ocean).

En realidad cada género de juego tuvo un representante digno y cada vez aparecían títulos con guiones más complejos, como *Everyone's a Wally* (Mikrogen), *Pyjamarama* de David Perry de la actual Shiny Entertaiment o el futurista juego *Tau Ceti* (CRL). En este año apareció el primer *Shoot'em up* con la técnica de isometría de Ultimate, *Highway Encounter*, escrito por Costa Payani. Una versión del juego *Uridium* (Hewson) del C-64 se convirtió en el juego para Spectrum con el mejor scroll. De hecho, la nave del protagonista, aun en movimiento, siempre estaba en el centro de la pantalla; era todo lo demás lo que se movía con una suavidad increíble. Este año también vio el nacimiento del que quizás sea el mejor juego para ordenador personal, *Elite* de Firebird, originalmente escrito para los ordenadores BBC en 1984, que en sólo 30 Kb te introducía en un universo de aventuras y estrategia en 3D vectorial.

1986

Quizás el Spectrum fue el ordenador personal que más hondo caló en los españoles. Ya en el 84 unos chavales de 19 años vendieron más de 20.000 copias de su juego *Fred*. Después, fundaron Made in Spain y crearon un clásico *Sir Fred*. Ya anteriormente, Víctor Ruiz y su hermano Pablo escribían *Saimazoon, Babaliba y Profanation* y nació Dinamic (ya desaparecida), sin duda, la empresa emblemática del panorama español. Le siguieron Opera Soft con *Cosa Nostra*. Este año y el siguiente fueron pletóricos para el software español. Fuera de España se seguían desarrollando buenos títulos, aunque técnicamente el Spectrum no daba más de sí. Ya sólo era cuestión cuidar los gráficos, la jugabilidad o la historia de los juegos. Podemos encontrar majestuosas producciones como *The Great Scape* (Ocean), *LightForce* (FTL) o el magnífico



1

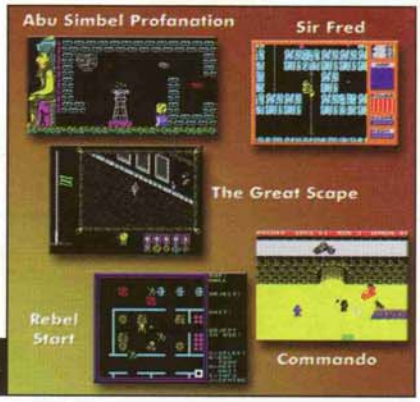
Año 1985. Ultimate Play The Game introduce la técnica Filmation con el juego Knight Lore.

2

Año 1985. Gran variedad y riqueza de juegos.

3

Año 1986. Primeros juegos con técnica de sprites y desarrollo de gran número de juegos de peleas y deportivos.



4

Año 1986. Se consagran todo tipo de géneros y aparecen juegos españoles de gran calidad.

5

Año 1987. El software español llega a lo más alto.

6

Las aventuras gráficas triunfaron con la llegada del PC.

7

Alone in the Dark introdujo el género "horror survival" a las aventuras gráficas.

Commando de Elite. Hubo un juego en este año que destacó siendo de un género poco agraciado en el Spectrum como son los *Wargames*; nos referimos al *RebelStart* de FireBird, que permitía la opción de dos jugadores. Al final del año triunfaron verdaderas maravillas como *Batman* (Ocean) o *La Armadura Sagrada de Antiriad*, que destacaba por la calidad de movimientos de los personajes.

1987

Dinamic triunfaba con gráficos cuidados al mínimo detalle y creando joyas como *Army Moves*, *Freddy Hardest*, *Game Over* o *Camelot Warrior*. El software español, por primera vez, arrasaba en Europa.

En general se hacían menos juegos, pero con mucha más calidad y mayor extensión, debido en parte a la aparición del Spectrum +2 con 128 Kb. Además, se desarrollaban muchas segundas partes de glorias pasadas.

Este fue el año de los juegos en España, pero quizás el éxito llegara un poco tarde. A partir de 1988 aparecieron los ordenadores de 16 bits y el Z80A no pudo competir con los 68000 de Motorola o los *Intel*. Sólo pudimos esperar algunas versiones de juegos de 8 bits pasados a 16 bits.

LA LLEGADA DEL PC

A partir de ahora, nos centraremos en el estudio de los juegos para PC. Haremos un breve repaso por los juegos más importantes en cada género. Antes, comentaremos el comienzo del PC como herramienta lúdica y los primeros juegos que se podían ver en él.

La historia del PC como plataforma lúdica comienza con su paso de las oficinas a los hogares, cuando no tuvo más remedio que adaptarse a las exigencias del público. El uso del PC se generaliza y a pesar de sistemas operativos visuales más avanzados y arquitecturas con posibilidades multimedia como el de Macintosh, Atari o los primeros Amiga, el pobre sistema operativo MS DOS triunfa en todo el mundo sólo por el hecho de la expansión de Microsoft.

Los videojuegos eran difíciles de implementar en un sistema donde el sonido no pasaba de pitidos del altavoz interno o unos gráficos en blanco y negro. Sin embargo, existía la ventaja de una mayor capacidad de almacenamiento de información, primero con los discos flexibles y posteriormente con discos duros y de una cantidad de memoria cada vez más ampliable. Los fabricantes de hadware se decantaron por el PC y pronto hacían su aparición tarjetas de vídeo y sonido para dotar al PC de color y de posibilidades musicales. Aunque, por entonces, existían lenguajes de programación bajo MS DOS como BASIC, PASCAL o C, éstos no serían suficientes para aprovechar todas las capacidades de los 16 bits. Los desarrolladores de videojuegos no tenían más remedio que seguir programando en código máquina. El problema de programar el PC era el continuo reciclaje que tenían que sufrir los programadores para adaptarse al incesante avance de los microprocesadores de Intel. No obstante, este avance ayudó a romper la barrera de los 640 Kb de memoria con el 80286. Poco a poco se convirtió en la plataforma ideal para programar en código máquina, debido a su rendimiento, capacidad de almacenamiento y de memoria y a los numerosos y potentes dispositivos externos como las impresoras matriciales o el almacenaje masivo. Se hizo muy común el desarrollo de juegos en estos sistemas para luego pasarlos a equipos de 8 bits.

En el próximo número...

... seguiremos hablando de los juegos para PC.

Cuestionario Videojuegos 5

Preguntas

1. Define una estructura de datos para la nave enemiga del juego en Blitz3D. Se necesita manejar su posición en un espacio 3D, su estado (tipo de acción) y el tipo de gráfico que tiene.
2. Realizar en Blitz3D las operaciones necesarias para establecer un modo gráfico de 800 x 600 y 16 bits de color en modo ventana. También se debe controlar si este modo es soportado por el ordenador.
3. Para aumentar el rendimiento de un juego se puede reducir el rango de visualización de la cámara, pero ¿cómo se puede disimular el corte en el campo de visión que este método genera?
4. ¿Cómo se puede texturizar un terreno de grandes dimensiones con texturas pequeñas?
5. En Paint Shop Pro, ¿cómo podemos mover o seleccionar un vector o forma geométrica?
6. ¿Qué significa "rasterizar una capa" y para qué se utiliza este procedimiento?
7. Hemos instalado en nuestro ordenador una tarjeta de sonido nueva. En el programa GoldWave, ¿cómo podemos seleccionar este nuevo dispositivo?
8. En nuestro programa GoldWave hemos grabado una canción, ¿cómo podemos indicar el comienzo del estribillo?
9. ¿Cuál es el método más rápido para crear una nueva pista en CUBASE?
10. ¿Con cuál herramienta y dónde podemos realizar una subida de volumen en un trozo de la canción?

Respuestas al cuestionario 4

▷ **1.**
```
If ...Then
Select ... Case
```

▷ **2.**
```
Repeat ... Until
Repeat ... Forever
While ... Wend
For ... Next
```

▷ **3.** El protagonista principal es el elemento o personaje del juego que el jugador controla directamente.

▷ **4.**. Antes de crear a nuestro protagonista, debemos diseñar su aspecto, su formar de actuar o los tipos de armas que utiliza. Todo esto es necesario para organizar el trabajo y obtener mejores resultados finales.

▷ **5.** Seleccionar F1, Mover F2, Rotar F3 y Escalar F4.

▷ **6.** Pulsar sobre el icono "Reemplazar Color" y elegir el color que queremos reemplazar. Luego en la paleta de colores elegimos como color de fondo el que reemplazará al seleccionado. Conmutamos el color de fondo por el de tinta y pintamos.

▷ **7.** Seleccionando el número de veces en la casilla de la opción "loop" del la ventana flotante de control.

▷ **8.** Mediante las marcas de selección. Desplazando el ratón con el botón izquierdo pulsado movemos la marca de comienzo de selección y lo mismo, pero con el botón derecho, la de final de selección.

▷ **9.** "M" significa si la pista está mutada o no.
"C" indica el tipo de pista.
"Chn" el canal MIDI de salida.

▷ **10.** Reproducir: "Enter numérico".
Parar reproducción: "Espacio".
Ir a la posición del localizador izquierdo: "1".
Ir a la posición del localizador derecho: "2".
Ir al principio de la canción: "5".

Contenido CD-ROM 5

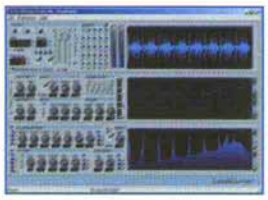
▶ AUDIO

■ ScrewLab 3.2
Interesante utilidad que te permitirá grabar música desde internet en formato mp3 y hacer remezclas.

■ Tunchy 1.0.
Programa gratuito con el que podremos mezclar composiciones en distintos formatos.

■ BeatBurner 1.1.
Potente sintetizador en tiempo real capaz de producir bucles de música únicos.

■ Blaze Media Pro 2002n



Reproduce, edita y convierte ficheros en cualquier formato de audio o vídeo.

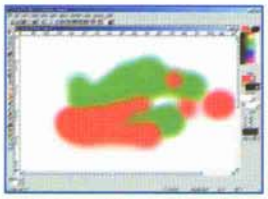
■ AdvaPlay 2.0.
Prueba este avanzadísimo reproductor de audio que sin duda te sorprenderá, ya que tiene multitud de opciones interesantes.

▶ DISEÑO 2D

■ LView 2002 Pro
Procesador de imágenes con múltiples opciones de edición, catalogación, visualización, etc...

■ Capture Screen Studio 3.0
Esta aplicación te será muy útil para capturar pantallas satisfactoriamente.

■ Image Maker 1.0



Crea tus imágenes, procésalas, edítalas y mézclalas con este potente programa.

■ WizardBrush 3.85
Software para realizar ilustraciones y que contiene además un montón de utilidades.

■ TextArt 1.0.
Crea tus propias fuentes, para que tu juego sea absolutamente personal.

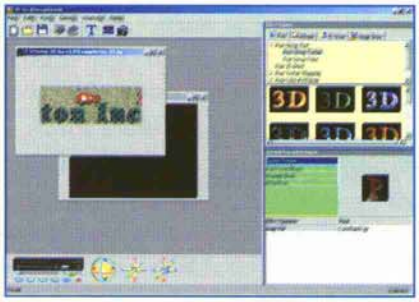
■ DemaResizer
Si quieres agrandar imágenes sin que éstas pierdan demasiada calidad y salgan muy pixeladas, esta herramienta te entusiasmará.

▶ DISEÑO 3D

■ Anim8or 0.8
Programa gratuito para realizar sencillas animaciones en 3d, muy sencillo de usar y bastante potente.

■ Direct 3d Wizard 1.4
Crea escenarios en tres dimensiones. Puedes editar, rotar, cambiar el enfoque, etc...

■ 3d Go 3.0



Potente aplicación con la cual podrás crear toda clase de texto animado.

■ HexaGreat 3d
Versión en inglés del interesante programa japonés, que nos ayudará con su sencillez para modelar polígonos.

■ ArCon 5.0
Con este software podrás transformar imágenes en 2d en modelos en tres dimensiones.

▶ PROGRAMACIÓN

■ Game Book 1.1.0
Archiva y mantén el código de tus juegos de cualquier plataforma.

■ Microsoft DirectX drivers 8.1
Drivers gratuitos de Microsoft para DirectX.

■ Role Playing Assistant 5.3
Crea tus propios juegos de rol sin necesidad de ser un programador experto.

■ Ghost Installer Free Edition

Gracias a este programa gratuito podrás crear un instalador para tu juego.

■ Ethos Games Basic
Desarrolla un proyecto completo de programación, depura el código, y todo fácilmente.

■ PureBASIC 3.0.
Compila en lenguaje BASIC de un modo sencillo y rápido.

■ Liberty BASIC for Windows 3.01
Gracias a esta potente herramienta, podrás escribir programas orientados a Windows.

▶ JUEGOS Y EMULADORES

■ Deus Ex
Excelente demo del popular juego que nos muestra un futuro terrible y no tan lejano.

■ The curse or Monkey Island

Demo del maravilloso juego de Lucas que tantos ratos de diversión nos proporcionó.

■ Emulador del Sinclair QL junto con un montón de roms.

■ Zone of Figghters
Nueva oportunidad para hacerte con la segunda entrega.

▶ VÍDEO

■ Muvee Autoproducer Cobalt 2.1
Creación de vídeos con un sencillo entorno para editarlos fácil y rápidamente.

■ Hyplay 1.2.0326.1
Reproductor de vídeo que soporta una multitud de formatos.

■ Movie Phile 3.5
Con este catalogador podrás tener mucho más control sobre tus archivos de vídeo.

■ MorphMan 3.0
Crea transiciones y morphings en tus vídeos con este práctico programa.

▶ EXTRAS

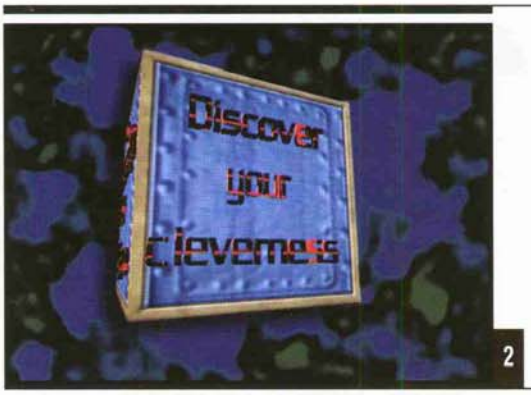
Todos los ejemplos e imágenes vistas en este número.

Diseño de los elementos de un juego III: presentación, menús y tutoriales

En la anterior entrega diseñamos el entorno del juego con todos sus elementos. Nos queda plantear los menús, el aspecto que tendrá la pantalla de juego y algunos de los elementos necesarios para la correcta terminación de nuestro trabajo, como la presentación y los tutoriales de ayuda.

(■) PRESENTACIÓN

Todo juego tiene que tener una presentación por pequeña y simple que sea.
Generalmente, se compone de varias partes que van desde mostrar los logotipos del distribuidor y desarrollador hasta el argumento del juego. Se pueden realizar presentaciones que sean verdaderas películas de animación o simplemente mostrar una pantalla fija con los logos y un texto argumentativo. La cuestión es que el futuro jugador sepa a qué clase de aventura se enfrenta. Generalmente, el argumento del juego se muestra en esta parte y aunque su finalidad sólo sea ésa, cumple un papel muy importante en la captación del interés del usuario, que tiene que perder algún tiempo antes de poder jugar. Debido a esto, es conveniente que la presentación pueda ser cortada a voluntad haciendo clic con el ratón o pulsando una tecla.



2

En la imagen se muestra una animación de presentación realizada con el propio motor 3D del juego.

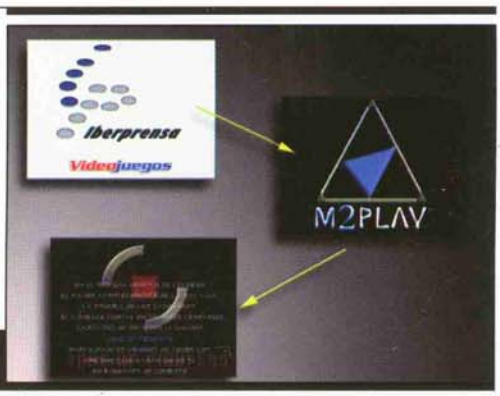
(■) LOGOTIPOS

Pero centrémonos en cada una de las partes que forman la presentación, empezando por los logotipos. Estos elementos sirven para colocarle al juego la "etiqueta de origen". Se muestran los logotipos de quien lo distribuye o de quien lo ha realizado y por último el propio logo o título del juego. Hay que tener en cuenta que el orden de aparición es importante. En primer lugar debe aparecer el distribuidor, seguidamente el desarrollador o entidades colaboradoras y por último la presentación del juego y su argumento.

(■) MOSTRAR EL ARGUMENTO

Por lo general, existe una clara división entre presentar a los artífices del juego y la descripción de éste. La mayoría de las distribuidoras y desarrolladoras tienen su forma de mostrarse al público, bien con una animación de su logo o con una pantalla fija y en muchas ocasiones utilizan diferentes métodos gráficos. Lo que sí tenemos que darle importancia es a la presentación del juego en sí, es decir, de su argumento. Desde un cortometraje de animación hasta un texto, todo vale, pero la calidad de esta parte del proceso dará una impresión mejor del juego. Lo primordial es contar de qué va. Dónde se desarrolla la acción, quiénes son los malos y los buenos y qué misión debe cumplir el jugador.
Actualmente se realizan verdaderas joyas de la animación para presentar un juego y que se utilizan también como reclamo publicitario en su comercialización, al igual que se hace con las películas del cine. Para los más modestos, es suficiente con cumplir su cometido aunque no sea una gran producción, sin embargo hay que tener siempre en cuenta mantener, ante todo, la calidad aun siendo una sola pantalla.

Si decidimos realizar una pequeña animación para presentar nuestro juego, hay varias formas de hacerlo. Se puede utilizar tanto el propio motor gráfico del juego como los modelos, con lo que perderíamos la espectacularidad que ofrece las animaciones creadas con paquetes 3D. Sin



1

Un ejemplo del flujo en una presentación aplicada a "Zone of Fighters".

embargo, esta técnica es ideal para largas presentaciones y mostrará al jugador el aspecto real que tendrá el juego. Por otro lado, la creación de una película de animación conlleva la utilización de aplicaciones 3D especializadas, lo que significa un mayor desembolso. Se consume más espacio en el disco y se deben utilizar técnicas de compresión de vídeo para que su reproducción sea posible en gran variedad de sistemas, lo que no siempre da el resultado deseado.

Si en nuestro diseño sólo contemplamos algunas pantallas fijas mostrando un texto con el argumento (como en el caso de "Zone of Fighters"), podemos añadir algún tipo de voz narrativa con música de fondo para crear más ambiente y en cierta manera obligar al jugador a leer todo el texto. Resumiendo, el argumento debe presentarse al usuario del juego de una forma atractiva, ordenada y con texto o voz claros y concisos.

TUTORIALES DE AYUDA

Estos elementos son importantísimos en el desarrollo de un juego. Los tutoriales son el manual de usuario y resultan necesarios para transmitir el procedimiento a seguir para sacar el mayor rendimiento al juego. Evidentemente, dependiendo del juego, los tutoriales serán de una forma u otra. Juegos de una temática de uso compleja, como los de estrategia o rol, suelen poseer tutoriales interactivos muy completos, normalmente incluidos en el menú principal. Otros incluyen niveles de aprendizaje o entrenamiento y los más simples, algunos textos explicativos.

La forma más corriente de implementar un tutorial es de forma externa, incluido en la instalación del juego. Podemos encontrar tutoriales para leer con procesadores de texto o en formato HTML para cualquier visualizador web. En todos debe aparecer una descripción del juego y de todos los elementos que lo componen, una relación de todos los controles disponibles para el jugador y por último la lista de requerimientos técnicos mínimos necesarios para el funcionamiento del juego.

Una vez dentro del juego también debemos procurar una opción que muestre la lista de controles y la misión a cumplir. Es muy normal que el jugador no lea las instrucciones del juego, y sin perder tiempo empiece a jugar sin saber siquiera cómo se maneja su personaje. Por eso tenemos que tener esta situación prevista y proporcionarle en cualquier momento la posibilidad de acceder a una ayuda breve durante la partida. Puede valer algún esquema o dibujo o simplemente una ventana de texto en la pantalla.

En "Zone of Fighters" diseñaremos un tutorial en HTML de forma externa y un panel gráfico dentro del juego que mostrará una relación de controles. Además, implementaremos una tecla para que cuando el jugador la pulse pueda ver de nuevo la lista de controles en cualquier momento de la partida y sin ningún tipo de interrupción.

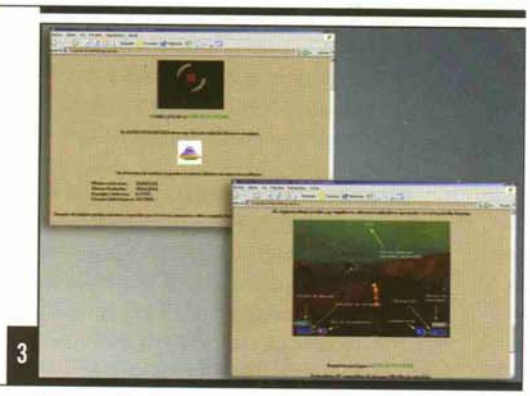
4

Es importante que el jugador pueda disponer durante la partida de un cuadro de ayuda con una descripción de los controles.

LOS MENÚS

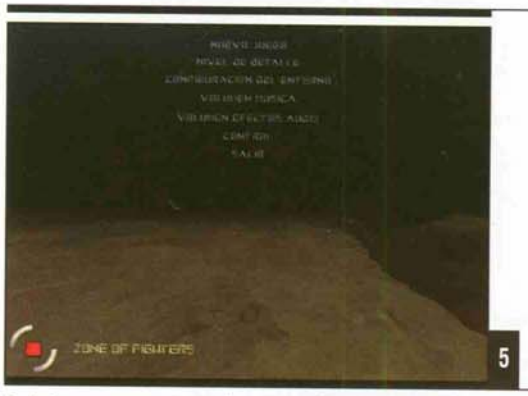
Con el menú entramos definitivamente en el juego. Está formado por numerosas opciones que a su vez conducen a otros submenús. El diseño de los menús es una parte importantísima y debe poseer la misma estética que el resto del juego, además de una interfaz sencilla de usar.

La configuración de las características del juego se realiza en este apartado y todo depende de la cantidad de opciones que le ofrecemos al usuario. Hay menús más o menos completos y con una



3

Los tutoriales en HTML son un formato válido para acompañar la documentación de un juego.



5

En la imagen se muestra el aspecto del menú de "Zone of Fighters".

estructura y capacidades que dependen mucho del tipo de juego. Pero, básicamente, podemos distinguir las siguientes partes en el menú de un juego: opciones gráficas, de sonido, de control y de juego.

(■) OPCIONES DE JUEGO

Dependiendo del tipo de juego podemos configurar multitud de aspectos. Lo más básico es, simplemente, empezar a jugar. Sin embargo, hay juegos que por sus características necesitan ampliar este apartado, desde la configuración del personaje (aspecto, nombre) o la dificultad a la elección del nivel de partida.

En "Zone of Fighters" en un principio se dará la posibilidad de elegir entre dos tipos distintos de terrenos, pero posteriormente podemos incluir la elección de mapas de combate creados con el editor de zonas o la generación aleatoria por parte del ordenador.

(■) OPCIONES GRÁFICAS

Debido a la gran cantidad de sistemas de diferente potencia gráfica, es necesario incorporar al juego una serie de opciones que permitan al usuario regular el rendimiento del juego. Principalmente se permite controlar la resolución de pantalla, la cantidad de colores, la elección de tarjeta gráfica u otros aspectos más técnicos como la configuración del entorno de juego como luces, sombras, etc.

En nuestro caso hemos preferido diseñar un menú aparte del menú principal del juego para elegir la resolución y potencia gráficas. Ya dentro del juego diferenciamos dos opciones, una para el nivel de detalle y otra para configurar el entorno. Esto es así dadas las características de "Zone of Fighters", del tipo de juego que es y de dónde se desarrolla. Al elegir "Nivel de detalle" entramos en otro submenú que nos permite seleccionar las siguientes opciones:

- **Detalle del terreno.** Bajo, medio y alto. Aquí elegimos la cantidad de polígonos que tendrá el terreno de juego. Por defecto colocamos el detalle en un nivel medio.
- **Sombreado del terreno.** Podemos activar o desactivar este efecto llamado *shading* y que produce sombra en el terreno dependiendo de la posición de la luz. Por defecto se encuentra activado.
- **Antialisado.** Esta opción permite activar o desactivar el suavizado de bordes de los polígonos. Inicialmente desactivado.
- **Rango de visión de la cámara.** Bajo, medio y alto. Con esta opción podemos ampliar hasta dónde es capaz de ver la cámara. Por defecto activamos un rango medio.

En el submenú "configuración del entorno", definimos todo el aspecto que tendrá la ambientación del juego. Diseñamos las siguientes:

- **Activación o no de niebla.** Por defecto está activada. En caso de quitarla del entorno debemos situar el rango de visualización más allá del límite de las paredes de la urna para evitar los cortes en la vista.
- **Activación o no del cielo y del reflejo del agua.** Por defecto están desactivados.
- **Activar el paso del día a la noche y viceversa.** Por defecto está activado y consiste en una disminución de la intensidad de la luz ambiental a la vez que se rota en su eje X.
- **Cantidad de decorado.** Bajo, medio, alto. Por defecto colocamos una cantidad media. Esta opción sólo tiene efecto cuando se elige la opción de juego de zona de combate aleatoria, y consiste en colocar una cantidad baja, media o alta de objetos de decorado sobre el terreno.
- **Activar o no el terreno dinámico.** Por defecto está activado y su finalidad es permitir la creación de agujeros producidos por los impactos de las bombas de minifusión.
- **Activar o desactivar decorado interactivo.** Por defecto está también activado y consiste en que los disparos dejen o no huella en los edificios en forma de manchas negras.

(■) OPCIONES DE SONIDO

En este apartado simplemente se permite el cambio de volúmenes del audio del juego, desde la música hasta los efectos sonoros como disparos, explosiones o voces.

Para nuestro juego, diseñamos la posibilidad de elegir el volumen de la música y de los efectos de audio en niveles bajo, medio y alto.

(■) OPCIONES DE CONTROL

Esta parte del menú es muy importante. Aquí se informa al jugador de cuáles son los controles para manejar el juego, además de permitir elegir entre diferentes dispositivos como teclado, ratón, joystick o *gamepad* (mando de juegos).

Es muy común permitir también el cambio por parte

6

Las opciones de configuración gráficas deben ser todo lo variadas posible.

del jugador de las teclas de control comúnmente llamado "configuración de teclas". "Zone of Fighters" tiene unos controles fijos, no pueden ser configurables, así que mostraremos un panel gráfico con la lista de controles. El juego sería capaz de jugarse perfectamente sólo con el ratón, aunque habrá más controles para manejar las vistas de cámara y la activación del escudo y camuflaje.

ASPECTO DE LA PANTALLA DE JUEGO. INDICADORES

Los indicadores juegan un papel importante, ya que su misión es informar al jugador de todo lo concerniente al desarrollo de la partida y al estado de su personaje. También se incorporan otros elementos como sistemas de guía o ayuda.

Pueden ser todo lo variados que la imaginación pueda alcanzar y su aspecto tiene que ir en concordancia con el juego. Lo fundamental es que cumplan su misión de manera clara y concisa ya que multitud de ellos en la pantalla pueden incluso molestar al desarrollo de la partida.

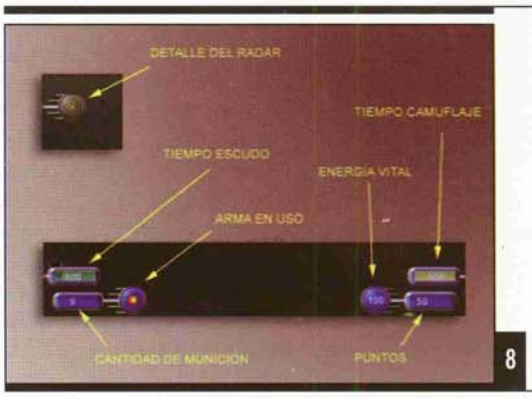
Vamos a diseñar para nuestro juego unos indicadores más o menos estándar y muy claros de distinguir. La idea es colocarlos en la pantalla de forma estratégica para que la vista de la acción sea lo más grande posible. Podríamos situar sólo números sobre la pantalla, pero seguramente pasarían desapercibidos y con el grueso de la acción serían prácticamente invisibles al jugador. Por eso, vamos a impresionar estas cantidades sobre algún tipo de gráfico que simule una pequeña pantalla de cristal. Podemos jugar con los colores y situación de estas pantallas para que el jugador las identifique rápidamente sin tener que desviar mucho la mirada.

Utilizaremos para estos indicadores un diseño geométrico y estilizado, sin muchos adornos. Estudiando las características del juego tendremos que informar al jugador con siete indicadores distintos situados a ambos lados de la pantalla en grupo de tres. El séptimo indicador será un radar, que muestre por medio de puntos de colores la situación del jugador (punto azul), de los demás participantes (puntos rojos) y de los voladores (puntos amarillos). Estará situada discretamente en la esquina superior izquierda de la pantalla y con forma esférica.

Colocamos en la parte inferior derecha el grupo de tres indicadores para la energía vital de la bionave, la cantidad de puntos conseguidos y el tiempo restante de camuflaje. En la parte inferior izquierda encontraremos un indicador con un gráfico del arma en uso y otro con la cantidad de munición de dicha arma y por último la cantidad restante de tiempo de escudo.

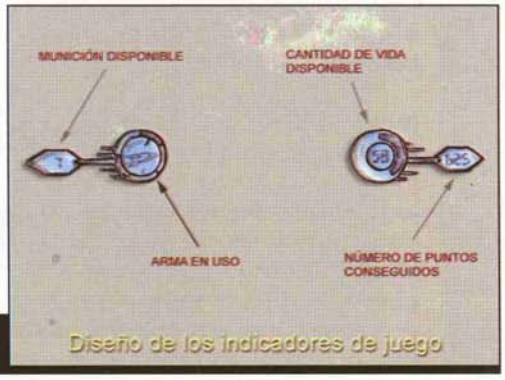
Otros tipos de indicadores son necesarios para "Zone of Fighters", como el punto de mira para la vista en primera persona y una cantidad numérica en la parte central superior de la pantalla que aparece cuando algún luchador, planta o animal es alcanzado por un disparo, indicando su nivel vital. Para el punto de mira se han diseñado dos tipos diferentes dependiendo del arma que se elija: uno alargado para la munición de bajo calibre y misiles (ya que la bionave dispara por cañones situados a los lados) y otro circular para el disparo de bombas.

Hasta aquí tenemos ya diseñado todos los aspectos que forman "Zone of Fighters". Podemos aplicar, fácilmente, todas las ideas expuestas a otros juegos y lo importante es saber que cuando se diseña un juego hay que pensar siempre en el jugador. Conseguiremos de esta forma que nuestro juego guste y sea aceptado por la comunidad.



8

Una descripción de los indicadores de pantalla del juego "Zone of Fighters".



7

Un ejemplo de diseño de indicadores de pantalla.

En el próximo número...

... empezaremos la programación de "Zone of Fighters", pero antes debemos prepararnos algunas herramientas que faciliten el trabajo de desarrollo, como visualizadores de modelos y terrenos, y plantearnos un editor de zonas de combate.

Grafismo 2D. La Tipografía

En el número anterior completamos la realización del logotipo y la pantalla de presentación de nuestro juego. Realizaremos en esta entrega nuestro propio tipo de letra, modificando una fuente compatible con Windows y, seguidamente, hablaremos un poco sobre los formatos 3D que utilizaremos en "Zone of Fighters".

Para nuestra fuente elegiremos de base el tipo "dungeon" que ya utilizábamos en la pantalla de presentación y cambiaremos el perfil de las letras desde la "A" hasta la "Z" mayúsculas para adaptarlas a nuestro diseño con el programa Font Creator.

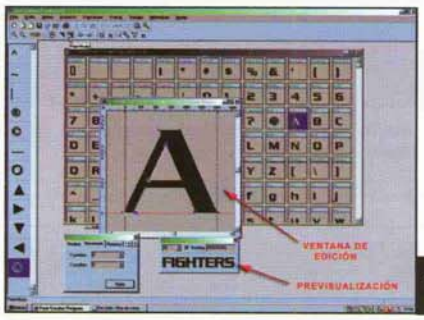
LA TIPOGRAFÍA

Los tipos de letras son muy importantes para reconocer nuestros productos y darles personalidad. Por ejemplo, en publicidad es conocido el poder que tienen para identificar una determinada marca. Por ello, se debe prestar atención en su diseño como una parte gráfica más del juego. Para construir un tipo de letra propio se pueden utilizar varias técnicas, como son dibujarlas o crear una fuente compatible con Windows. Dibujar un alfabeto, convirtiendo cada letra en un *sprite*, ha sido la manera tradicional de realizar una tipografía. Con este sistema es fácil generar una letra colorista y de formas muy variadas obteniendo resultados realmente originales. El inconveniente radica en su utilización dentro del juego ya que, al tratarse de imágenes, no resulta tan fácil imprimir un texto como lo haríamos utilizando una fuente *true type font* para Windows. Tendríamos que sustituir los valores ASCII por su correspondiente en imagen. Realizaremos un estudio de cómo funciona este procedimiento en uno de los tutoriales de programación de próximos números. Por el momento, en "Zone of Fighters" utilizaremos una tipografía compatible con Windows, ya que es un método más fácil de implementar y nos servirá para otros tipos de grafismos; por ejemplo, sustituyendo la letra del título del juego por la nueva que hayamos creado.

CREANDO NUESTRA PROPIA FUENTE

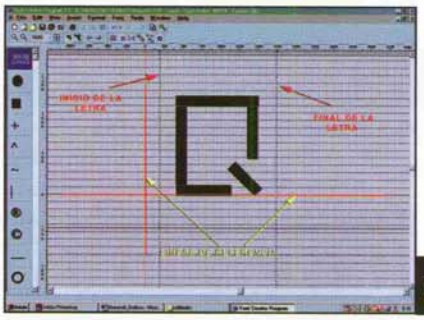
Una vez dentro del programa *Font Creator* tenemos que abrir la fuente *dungeon.ttf*. Para ello elegimos la opción "Installed Font" en "Open" del menú "File" o pulsando CTRL + I. Aparecerá una lista de tipos de letras. Elegimos la "dungeon normal (true type)". Una vez cargada, se mostrarán en una ventana todos los caracteres de la fuente con la tecla asociada a cada uno de ellos. Podemos apreciar también en la parte inferior una ventana de diálogo llamada "Preview". En ella veremos cómo van quedando nuestras letras, escribiendo en la línea de texto y eligiendo un tamaño. Por el momento, vamos a escribir "ABCDEF" con mayúsculas y un tamaño de 30. A medida que modifiquemos estas letras, veremos los cambios efectuados inmediatamente en esta ventana. Empezamos con la letra "A" mayúscula. Si pulsamos dos veces seguidas sobre ella activaremos la ventana de edición (Fig. 1).

Nos encontramos con la letra sobre una cuadrícula, delimitada por dos líneas discontinuas que representan el comienzo y el final global de la letra y dos ejes de color rojo (líneas de referencia) que indican la posición vertical y horizontal con respecto a la línea



1

Al pulsar dos veces sobre la casilla de la letra abriremos la ventana de edición.



2

Líneas de referencia en la ventana de edición.

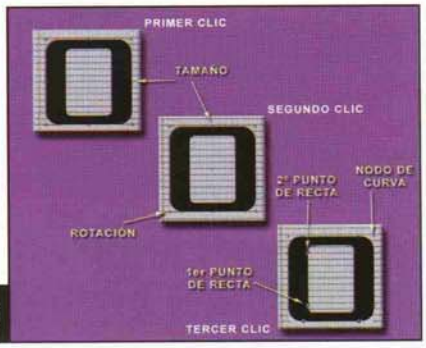
DEFINICIÓN

▸ TRUE TYPE FONT

***True Type Font* es simplemente un tipo de letra o fuente usado por todas las aplicaciones que corren bajo Windows. El archivo tiene la extensión .TTF y almacena una definición de los caracteres de dicha fuente.**

NOTA

Podemos modificar el aspecto, tamaño, etc. del grid o cuadrícula en "Tools/Grid Options", así como el color y la forma de las líneas de referencia.



3

Distintos modos de edición según el número de clics que hagamos con el ratón.

4

Aspecto definitivo de la letra "A".

5

Procesos para formar la letra "B".

NOTA

Es importante saber que moviendo las líneas discontinuas que marcan el principio y final de la letra, podemos crear más o menos espacio entre letras dentro de una palabra.

de texto. Sin dejar de pulsar sobre la letra y desplazando el ratón, la moveremos de posición en la línea de texto (Fig. 2).

Observamos que, al pulsar sobre ésta, se rodea de un cuadrado con seis cuadraditos o nodos, que nos servirán para modificar su tamaño. Si pulsamos dos veces, los nodos cuadrados se vuelven redondos y rectangulares; éstos nos servirán para transformarla y rotarla, y si pulsamos tres veces entraremos en la edición por puntos. Pulsando fuera de la letra volveremos al estado original de modo de contorno (Fig. 3).

Empezamos a modificar la letra "A" activando el "Point mode" o edición por puntos, pulsando tres veces sobre ella o activando la opción "View/Mode/Point". Los puntos de edición cuadrados representan comienzos y finales de líneas rectas y los puntos redondos nodos de control de curvas. Seleccionamos estos puntos redondos y los borramos. Pulsamos sobre cada uno de ellos con el botón derecho y elegimos "delete" en el menú emergente o simplemente pulsamos la tecla de suprimir. A continuación, desplazamos los puntos de rectas hasta formar la figura que se muestra en la imagen 4.

Pulsamos el icono de "Next Glyph" para pasar a la siguiente letra. Si en vez de hacer esto pulsamos dos veces sobre la siguiente letra, abriremos otra ventana de edición. Para la "B" realizaremos la misma operación que con la "A" con los nodos de curvas, es decir, los borramos. Podemos seleccionar más de un punto a la vez remarcándolo con el ratón o seleccionando uno a uno con la tecla SHIFT pulsada (Fig. 5).

Seguidamente, movemos los puntos de control cuadrados hasta obtener la forma que se muestra en el último cuadro de la figura 5. Para la letra "C" vamos a borrar la parte superior para quedarnos con forma de "L". Seleccionamos con el ratón todos los puntos de la parte superior de la letra y borramos. Posteriormente, eliminamos también todos los nodos de control de curvas. Para la parte superior, desplazaremos la barra horizontal de la librería, situada a la izquierda del programa, hacia la ventana de edición y le aumentaremos el ancho (Fig. 6).

La siguiente letra, la "D", es muy parecida a la "C", pero con una pequeña modificación; así que podemos copiar la "C" en el lugar de la "D" y modificarla. Pulsamos con el botón derecho sobre la casilla de la "C" y elegimos "Copy" (CTRL + C). Pulsamos sobre la letra "D" de igual forma y elegimos "Paste" (CTRL + V) (Fig. 8).

Luego seleccionamos la barra vertical de la letra hacia la derecha hasta obtener la forma de una "D" (Fig. 9).

La "E" es exactamente igual que la "C" pero con una barra horizontal en el centro. Entonces, podemos copiar la "C" en la casilla de la "E", pulsamos sobre la barra horizontal superior y hacemos CTRL + C y CTRL + V para hacer una copia de ella. Hay que tener en cuenta que la copia se realiza sobre el original; si pulsamos sobre ésta y desplazamos hacia abajo, veremos cómo movemos la copia. La "F" es igual que la "E" sin la barra horizontal inferior, y así sucesivamente con todas las demás letras. Las que son parecidas se copian y luego se transforman hasta obtener el resultado deseado. En la figura 10 se muestra la lista de todas las letras nuevas para "Zone of Fighters", desde la "A" hasta la "Z" mayúsculas, para que sirva de guía. A continuación, mostramos cómo realizar algunas operaciones importantes en el modo *edición por puntos*:

- **A.** Los puntos cuadrados de color verde son los primeros de su línea y los azules los consecutivos. Para convertir los puntos consecutivos en primeros y los que le siguen en secundarios hay que pulsar el botón derecho sobre el punto azul o secundario que queremos convertir y elegir la opción "First Point".

- **C.** Para crear un nuevo punto de edición en una línea, pulsamos en cualquiera de los puntos de dicha línea con el botón derecho y elegimos la opción "Add Point". Se creará un punto entre éste y el anterior, a no ser que realice sobre un punto primario (First Point), en cuyo caso el nuevo punto aparecerá a continuación.
- **D.** Para convertir un punto de línea (punto cuadrado) en curva, pulsamos con el botón derecho y elegimos la opción "Off Curve" en el menú emergente; para la operación contraria, en "On Curve".

Una vez que tenemos nuestra fuente terminada, podemos comprobar los resultados abriendo la ventana "Test Font" pulsando la tecla "F5". Antes de instalarla debemos guardarla con otro nombre, por ejemplo, "ZOF.TTF" con la opción "File/Save as" en el directorio "C:\Windows\Fonts". Para instalar la fuente elegimos la opción "Install" del menú "Fonts" (Fig. 11).

FORMATOS DE OBJETOS 3D

En el mundo de las aplicaciones 3D existen multitud de formatos de objetos diferentes. Sin embargo, todos tienen en común el ser utilizados para almacenar objetos y relacionarlos entre sí.
Prácticamente cada programa tiene su propio formato. Aun así, se han mantenido de uso común una serie de ellos que por su popularidad han creado estándares. Generalmente, este hecho se ha producido por el establecimiento en el mercado durante los comienzos del 3D de aquellos programas más usados. Es el caso de 3D Studio de Autodesk que implantó su formato .3DS en todo el mundo y hoy día es, con toda seguridad, el más usado en el desarrollo 3D, generalmente para el intercambio de objetos entre aplicaciones.

.3DS

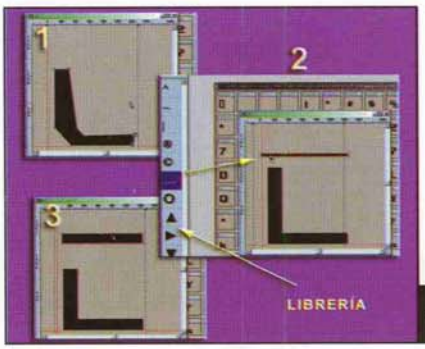
Es un formato sencillo de implementar en motores 3D debido a su estructura por trozos de datos almacenados jerárquicamente. Para el diseño de videojuegos es muy interesante porque, además de estar soportado por todos los programas de diseño 3D, puede llevar asignadas las texturas y la animación en el mismo fichero.

Blitz3D soporta este formato y aprovecharemos esta ventaja para utilizarlo en los objetos estáticos que aparecen en "Zone of Fighters": edificaciones, rocas, árboles y en la planta "Shaark". Aunque también podemos utilizar objetos con animación en .3DS, ya que Blitz3D es capaz de obtener la animación y la jerarquía almacenados. Si queremos utilizar la opción de colisión por polígonos del Blitz3D, resulta conveniente el uso de este formato en objetos animados para poder obtener unos resultados óptimos. Además, si en nuestra aplicación 3D asignamos las texturas al objeto y las incluimos en el mismo directorio que éste, no es necesario aplicarlas desde programación. Simplemente con cargar el objeto es suficiente para que se dibuje texturizado.

XOF O .X

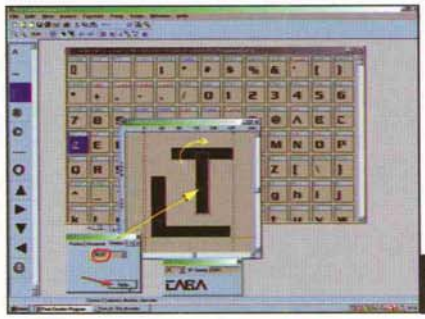
Hay un formato estándar en prácticamente todas las aplicaciones multimedia programadas con la API de Microsoft DirectX, nos referimos al XoF o .X.
Aunque este formato es el estándar de DirectX, no es soportado por la mayoría de los programas 3D. Comúnmente se trabaja con formatos como .3DS y luego son convertidos, con aplicaciones especiales, a .X. Lo que resulta atractivo en este formato es su utilización en motores gráficos, ya que se compone de una estructura en forma de plantilla que el programador puede utilizar y modificar a su gusto o bien usar plantillas base. En este formato también se utilizan árboles jerárquicos e igualmente almacena objetos, materiales y los keyframes de animación.

Blitz3D soporta este tipo de objetos y se comporta de la misma forma que el .3DS. Podemos exportar a este tipo de formato con la aplicación de modelado en



6

Procesos para formar la letra "L".



7

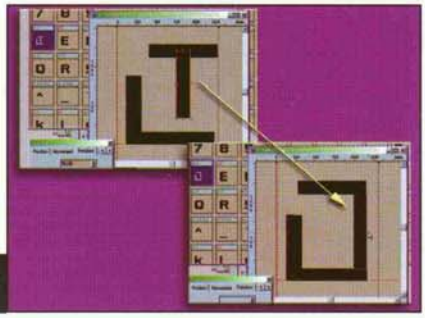
Procedimiento para rotar una porción de la letra.

8

Podemos copiar una casilla a otra para duplicar letras.

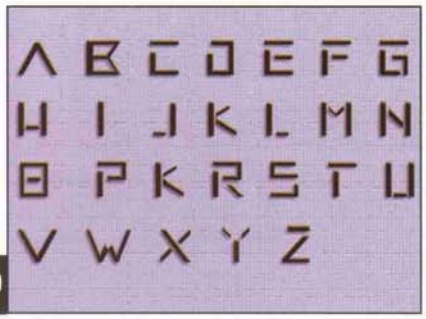
TRUCO

Hay otra manera de realizar las operaciones de movimiento, transformación, rotación, etc. de una forma precisa numéricamente en *Font Creator*, y es por medio de la ventana de diálogo "Transform", la cual se activa en "View/Toolbars/ Transformations" (Fig. 7).



9

Con pocas modificaciones se puede obtener una letra nueva a partir de otra.



10

Set de letras para Zone of Fighters.

11

Ventana de instalación para la nueva fuente.

TRUCO

Antes de empezar a crear tu propia fuente, fíjate bien en qué letras son parecidas entre sí para, de este modo, no trabajar dos veces. Podremos usar una misma letra como plantilla, y a partir de ésta formar todas las que se le asemejen.

baja Milkshape3D e importarlos desde el mapeador UV Lithunwrap , que utilizaremos en esta obra.

.OBJ

Este formato es un estándar de *Alias WaveFront Object* y es muy común su uso para pasar la geometría de los objetos de una aplicación 3D a otra. También es muy utilizado en visualizadores de objetos 3D. Este formato soporta tanto objetos poligonales (puntos, líneas y caras) como objetos de forma libre (curvas y superficies) y además puede ser almacenado en formato ASCII y binario, aunque el más utilizado es el primero de ellos.

Blitz3D no puede leer este tipo de ficheros, pero es el utilizado (además del .3DS) por *Character FX*, que utilizaremos para animar nuestros personajes. Así que tendremos que exportar a .OBJ o .3DS desde el modelador, para poder trabajar con los objetos desde *Character FX*, aunque en esta obra importaremos los objetos desde esta animación en formato .OBJ como un solo mesh.

.LWO

Este formato es el utilizado por la popular aplicación *3D LightWave*. Es un estupendo formato para almacenar animación, además, claro está, de objetos poligonales, curvas, superficies y texturas. Cada vez es más soportado por la mayoría de las aplicaciones más populares.

En nuestra obra no utilizaremos este formato porque no puede ser leído por Blitz3D directamente.

.MD2

Por último, este formato es una versión avanzada del .MDL impuesto por la empresa ID Software para su juego Quake. Este tipo de archivo contiene la geometría del modelo, información sobre la animación y las coordenadas y nombres de las texturas.

Se caracteriza por dar lugar a animaciones suaves gracias a su sistema de deformación de la malla que forma el objeto.

Este formato sí es soportado por Blitz3D y es una opción muy válida para utilizar objetos animados en nuestros juegos. El único inconveniente que presenta, es que la detección de un objeto animado en este formato no es muy precisa y sólo funciona con el tipo de colisión esfera a esfera. Este formato es el que utiliza *Character FX* para exportar las animaciones. Nos encontramos entonces con un dilema. Si utilizamos *Character FX* para nuestros propósitos, no tendremos más remedio que utilizar este formato en nuestro juego, con los problemas que conlleva. Sin embargo, la versión 1.76 de Blitz3D incorpora la utilización de un formato propio, el .B3D

.B3D

Este formato fue originando por el equipo creador del Blitz3D para la versión 1.76 de este lenguaje de programación. Su principal característica es que engloba geometría de objetos, texturas y frames de animación. Es decir, tiene la animación del formato .MD2 y las características geométricas de los formatos .3DS o .X.

Así que podemos poseer un objeto con animación por deformación de malla que puede ser detectado con todo tipo de colisiones (polígonos, cúbica, esférica).

Éste es el formato que utilizaremos en nuestro juego para los personajes animados como las plantas o animales. Para conseguir exportar un objeto .3DS o .OBJ al formato .B3D desde *Milkshape3D* o al mismísimo *Character FX*, están disponibles plugins para poder realizar esta operación en estas dos aplicaciones. Además podemos disponer también de un plugin para exportar a .B3D desde el propio *LightWave3D*.

En el próximo número...

... modelaremos nuestra bionave de combate con Milkshape3D.

Cómo desarrollar los efectos especiales de sonido (II)

Después de conocer cómo configurar Goldwave para la grabación y reproducción de audio, es el momento de entrar realmente en materia. Vamos a crear nuestro primer efecto de sonido partiendo de cero. Con él aprenderemos la aplicación de filtros y efectos de audio para obtener diferentes resultados con un rendimiento en tamaño y calidad aceptables.

NORMALIZAR Y ECUALIZAR

Cuando se graba un sonido desde una fuente exterior, no siempre se obtiene el aspecto sonoro deseado. En algunas ocasiones, suena demasiado agudo o demasiado grave, y en otras, con volumen muy bajo.

Podemos hacer una primera aproximación a la calidad que deseamos por medio de un par de funciones muy importantes: el normalizado y la ecualización.

Por medio de la normalización, podemos ajustar el volumen del sonido al límite máximo, antes de sufrir saturación. Por otro lado, ecualizando podemos cambiarle los rangos de frecuencia y variar de esta forma su tonalidad. Conseguimos así un sonido más rico y con cuerpo.

Cojamos una bolsa de plástico. Vamos a crear el sonido de un fuego. Será perfecto para comprender mejor el normalizado y la ecualización. Si manoseamos la bolsa con cuidado, notaremos que el sonido que produce se asemeja al del fuego quemando leña.

Preparamos el editor para grabar. Ajustamos el volumen del micrófono y deseleccionamos la opción "Loop" y "Ctrl key safety" de las propiedades de grabación. Creamos una plantilla vacía de 10 segundos de duración con calidad CD y grabamos.

Observamos que la grabación suena algo baja de volumen. Pues bien, vamos a ajustar su volumen al máximo sin saturar. En Goldwave tenemos varias opciones para amplificar el volumen en "Volume", en el menú "Effect". Para normalizar debemos seleccionar "Maximize", pinchar la opción "New maximum" y colocar el valor en 1.0. La opción "New RMS" es útil para igualar el volumen de varios ficheros de audio distintos, por ejemplo, de varias músicas para un CD (Fig. 1).

Una vez ajustado el volumen, vamos a darle más cuerpo al sonido por medio de la ecualización. Elegimos "Filter/Equalizer" en el menú "Effects". Entramos en un pequeño equalizador de siete bandas con frecuencias de 60 Hz (sonido grave) hasta 15 Khz

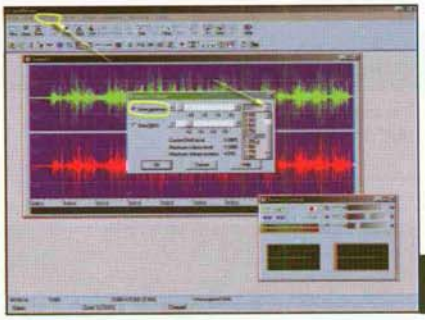
DEFINICIÓN

▸ DECIBELIOS

Los decibelios son una escala logarítmica de intensidad de la señal de audio; es decir, miden el volumen de un sonido.

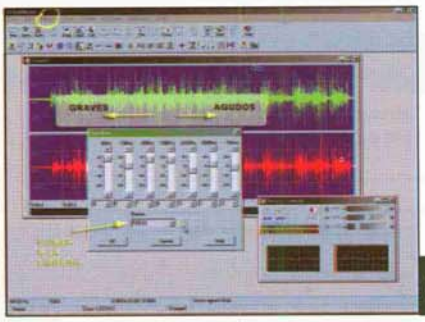
NOTA

Siempre es conveniente grabar y editar con la máxima calidad cualquier sonido, para posteriormente pasar a la resolución deseada. Con este sistema, conseguiremos unos mejores resultados.



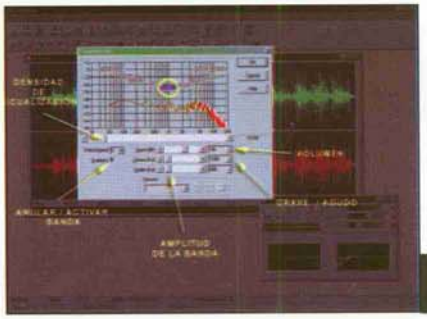
1

Mediante la función "Maximize volume" podemos obtener la normalización de nuestro sonido.



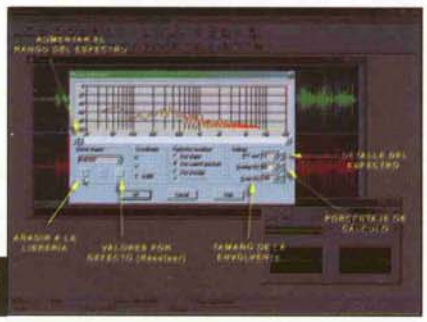
2

Descripción básica del ecualizado.



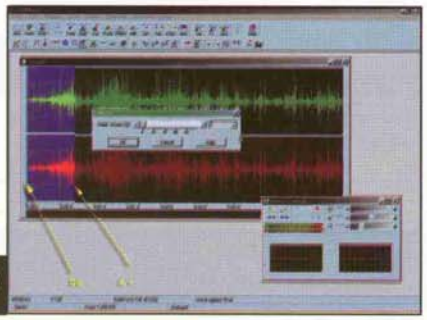
3

Descripción de la ventana del ecualizador paramétrico. (Imagen incluida en el CD-ROM).



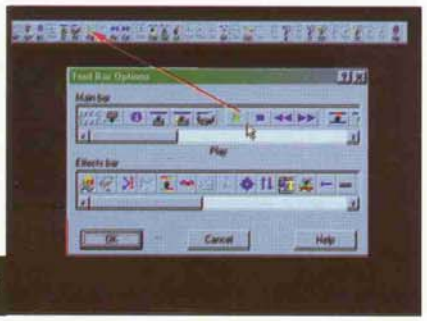
4

Descripción de la ventana del proceso de reducción de ruido.



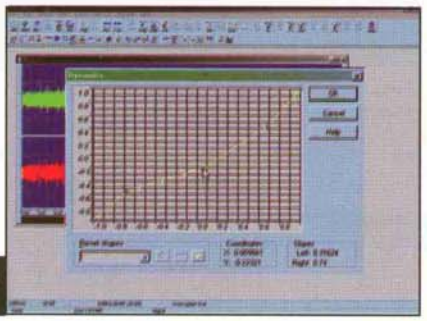
5

Mediante la función "Fade in" podemos realizar una subida de volumen.



6

A través de la opción "Options / Tool Bar" puedes elegir qué iconos se mostrarán o cuáles no, en la barra de herramientas y de efectos.



7

Mediante el efecto "Effects /Dynamics" podemos alterar el mapeado de amplitud de una selección a lo largo del tiempo.

(sonido agudo). El volumen de cada una de estas frecuencias (bandas) va desde -20 decibelios hasta 10 Db. Por ejemplo, si aumentamos a 10 Db la banda de 15 Khz, estamos subiendo al máximo la frecuencia más aguda del sonido (Fig. 2).

Un sonido con demasiados graves puede sonar sucio y sin claridad. Si, por el contrario, tiene demasiados agudos puede aparecer ruido. Disponemos de otro ecualizador mucho más flexible, potente y preciso: el paramétrico. Lo podemos encontrar en "Effects/Filter/Parametric EQ". Su función es la misma, la diferencia es un mayor control sobre las bandas y el rango de frecuencias y volúmenes que podemos configurar repartidos en 30 bandas (Fig. 3).

APLICACIÓN DE EFECTOS

Aplicando todo lo que hemos aprendido, vamos a crear un disparo a partir de una palmada. Prepara el editor para grabar. Crea una plantilla nueva de 5 segundos de duración con calidad CD. Empieza a grabar dando una palmada seca a unos 20 centímetros del micrófono para no saturar la grabación.

Una vez realizada la grabación, debemos eliminar el espacio vacío del comienzo y la coletilla de audio inservible del final.

NOTA

Recuerda que la marca de principio de selección se mueve con el botón izquierdo del ratón, y la de final de selección, con el botón derecho.

DEFINICIÓN

▸ ENVOLVENTE

La envolvente representa la dinámica de un sonido, es decir, los cambios que sufre en la frecuencia y el volumen.

Los señalamos con las marcas de posición de selección y borramos con CTRL + X. Aunque la grabación parezca tener un buen volumen, siempre es conveniente normalizarla.

Una vez normalizado el volumen, podemos quitar un poco de ruido con un filtro de reducción de ruido "Noise reduction", situado en "Effects/Filtres". Podemos descubrir una gráfica del espectro del sonido con la frecuencia en el eje X y los decibelios en el Y, además de una serie de opciones. Modificando las opciones en "Settings", podemos conseguir buenos resultados ya que determina las características del espectro. El detalle de éste lo determina la opción "FFT size", así que lo situamos en 12. La opción "Overlap" determina el porcentaje de cálculo del espectro. Le asignaremos un valor del 80%. Y con "Scale" podemos modificar el tamaño de la envolvente del sonido, con lo que a valores altos borramos más audio del sonido, así que los situamos en 100 % (Fig. 4).

A continuación, vamos a reducir el tono para obtener un sonido más alargado y grave. Elegimos la opción "Pitch" en "Effects". Deseleccionamos la opción "Preserve lenght" para no mantener la misma longitud. Activamos "Semitone" y elegimos el valor -12. Realizamos la operación dos veces, es decir, un semitono a -24. Para reducir el tiempo de ataque del comienzo, vamos a aplicar un efecto de menos a más volumen llamado "Fade in". Seleccionamos desde el principio hasta un tercio del sonido y elegimos la opción "Effects/Volume/Fade in" (Fig. 5).

Haremos lo mismo, pero con la operación contraria, seleccionando el final del sonido y aplicando "Effects/Volume/Fade out". Podemos aplicar cuantos efectos especiales nos apetezca hasta obtener el sonido que deseemos, por ejemplo, un poco de eco o un efecto *flanger*.

Manejo de texto e imágenes con Blitz 3D

El manejo de texto e imágenes es fundamental para el desarrollo de cualquier juego o aplicación multimedia.

Por ejemplo, las aventuras gráficas o las conversacionales se basan en estas dos cualidades. Aprenderemos a imprimir en pantalla y a controlar cadenas de texto, así como a cargar y controlar ficheros de imágenes.

IMPRIMIR TEXTO EN PANTALLA

La instrucción más básica para imprimir texto en pantalla es "Print":

```
Print "Aquí escribimos un texto"
```

Coloca el cursor en la posición (0,0) e imprime el *string* y luego pasa el cursor a una nueva línea. Si no queremos mover el cursor de línea debemos utilizar "Write":

```
Write "Aquí escribimos un texto"
```

Puede ocurrir que nos interese imprimir en cualquier parte de la pantalla. Para ello podemos utilizar la instrucción "Locate", la cual coloca el cursor en las coordenadas deseadas:

```
Locate Coordenada Horizontal,
Coordenada Vertical
Locate 10,20
Print "HOLA"
```

NOTA

Debemos saber que las instrucciones "Print" y "Write" escriben el texto en el búfer frontal ("FrontBuffer").

Existe, sin embargo, una función que engloba a las anteriores, se trata de "Text", que coloca y escribe un *string* en la pantalla gráfica:

```
Text Coordenada Horizontal,
coord. Vertical,"Texto",
centrado sX (true/false),
Centrado Y (true/false)
Text 200,100,"Aquí escribimos
un texto", true, falsee
```

Las dos últimas opciones se utilizan para centrar horizontal o verticalmente el texto.

Tanto a "Print", "Write" o "Text" podemos añadirles texto con el símbolo "+":

```
nombre$="Darío "
apellido$="Medina "
edad=1
Locate 100,10: Print nombre$+
apellido$+"tiene "+edad+" año"
Text 100,30, nombre$+
apellido$+"tiene "+edad+" año"
```

Resultado de ambos: Darío Medina tiene 1 año (Ver "ejemplo1.bb").

UTILIZANDO FUENTES DE LETRAS

En Blitz3D tenemos acceso a los tipos de fuentes instalados en Windows a excepción de las fuentes de símbolos como "Webdings". Para cargar una fuente utilizamos la función "LoadFont":

```
LoadFont ("nombre de la fuente",
altura true/false, negrita
true/false , itálica true/false,
subrayado true/False).
```

Y para poderla usar debemos activarla con *SetFont variable,* que contiene la fuente. Una vez usada la fuente se puede eliminar de memoria con la función:

```
FreeFont variable que contiene
la fuente.
```

En el "ejemplo1.bb" (CD-ROM, apartado "extras") podemos observar el funcionamiento de estas funciones.

MANEJANDO CADENAS DE TEXTO

Para empezar, comentaremos algunas pequeñas funciones útiles que siempre es bueno saber para optimizar mejor nuestro código.

Convertiremos nuestro texto a mayúsculas y a minúsculas con las instrucciones "Upper" y "Lower":

```
nombre$="Darío"
Print Upper(nombre$)
```

Resultado: DARÍO

```
nombre$="DArío"
Print Lower(nombre$)
```

Resultado: darío

Eliminamos los espacios posibles al principio y final de una frase con "Trim":

```
nombre$="    Darío    "
apellido$=" Medina "
Print Trim(nombre$)+ apellido$
```

Resultado: Darío Medina

Si queremos cambiar el tamaño de un *string*, podemos utilizar " Lset " y " Rset". Generalmente se utilizan para justificar el texto a la izquierda o a la derecha.

```
frase$="Mi coche verde "
frase$=Lset(frase$, 20)
Print ">"+frase$+"<"
```

Resultado:
>Mi coche verde <
"Mi coche verde" tiene 15 caracteres + 5 espacios = 20

```
frase$="Mi coche verde "
frase$=Lset(frase$,5)
Print ">"+frase$+"<"
```

Resultado: >Mi co<
"Mi coche verde" tiene 15 caracteres , sólo cogemos 5.
"Rset" funciona igual pero empieza a contar desde la derecha:

```
frase$="Mi coche verde "
frase$=Rset(frase$,20)
Print ">"+frase$+"<"
```

Resultado:
> Mi coche verde<
La instrucción "Replace" nos permite reemplazar los caracteres de un string por otro. Por ejemplo, si en la frase "Guadalajara tiene 6 a" queremos cambiar las "a" por "e", escribiríamos:

```
frase$="Guadalajara tiene 6 a"
Print frase$
Print Replace(frase$, "a", "b")
```

Resultado: Guedelejere tiene 6 e
Para encontrar la posición de un carácter o varios dentro de una frase se utiliza la instrucción "Instr":

```
Posicion = Instr (caracteres
a buscar, frase dóonde buscar,
posición de comienzo de búsqueda)
frase$="Guadalajara"
Print "La posición es: " +
Instr ( "a","frase$, 4)
```

DEFINICIÓN

▸ CÓDIGO ASCII
El código ASCII es un convenio que se adoptó para asignar un valor numérico a cada carácter.

Resultado: La posición es: 5
Empezamos a buscar la "a" a partir del cuarto carácter de la frase (la "d"), y obtendremos la posición de la primera "a" que encuentre a partir de ahí. Si no encuentra nada, devolverá el valor 0.
Para contar el número de caracteres de una frase (espacios incluidos) podemos utilizar la función "Len":

```
frase$="  hola"
Print  Len (frase$)
```

Resultado:
6 (dos espacios y hola)
Para encontrar caracteres dentro de una frase disponemos de tres instrucciones "Left ", "Right" y "Mid". Por ejemplo, queremos obtener la palabra "Mi" y la palabra "verde" de la frase "Mi coche verde":

```
frase$ = "Mi coche verde"
palabra1$ = Left (frase$,2)
palabra2$ = Right (frase$,5)
Print frase$ : Print palabra1$
+" "+ palabra2$
```

Resultado: Mi coche verde
Mi Verde
Obtenemos los 2 caracteres más a la izquierda ("Mi") de la frase y los 5 más a la derecha ("verde")
Imaginemos que deseamos finalizar nuestro programa cuando el usuario introduzca el símbolo "*" en el ordenador.
La instrucción "Mid" nos puede ayudar en ello, ya que recorre un *string* en busca de uno o varios caracteres determinados.

```
For n = 1 to Len(entrada_usuario$);
recorremos la frase escrita por
el usuario en busca del asterisco
  If Mid(entrada_usuario$,"*"n,1)=
"*" then End
Next
```

Si queremos detectar por ejemplo dos caracteres, por ejemplo, "no", lo sustituiríamos por:

```
If Mid(entrada_usuario$,n,2)=
"no" then End
```

Recorremos la frase introducida por el usuario. Si a partir de la posición "n" hay 2 caracteres que coinciden con "no" finalizamos el programa.
Obtenemos los 2 caracteres más a la izquierda ("Mi") de la frase y los 5 más a la derecha ("verde")
En el "ejemplo3.bb" incluido en el CD se muestra un ejemplo útil del uso de estas instrucciones. Observad el uso de la instrucción "Right" para aislar la extensión de los ficheros y "Mid" para detectar el "." que separa la extensión. No es necesario entender, por el momento, los comandos referentes al uso de ficheros. Dedicaremos un capítulo del curso especialmente a este tema.
Para terminar, una función que nos permitirá repetir un número determinado de veces un *string*: "String":

```
String (frase a repetir,
numero de veces que se repite)
```

FUNCIONES DE CONVERSIÓN

En ocasiones, necesitaremos realizar operaciones con la tabla ASCII. Por ejemplo, saber cuál es el código numérico de un determinado carácter o viceversa. Para obtener este tipo de información disponemos de "Asc" y "Chr".
Queremos saber el código ASCII de la letra "A": *Print Asc("A")*.
O lo contrario, el carácter correspondiente al código ASCII 65: *Print Chr(65)*.
Y por último, también podemos convertir un valor numérico en un *string* con la instrucción " Str ":

```
numero=25
numero2$=Str(numero)
Print numero + numero
Print numero2$ + numero2$
```

Resultado: 50
2525
(Ver "ejemplo4.bb")

FUNCIONES DE IMAGEN

En los juegos es muy común el uso de imágenes para ilustrar o crear fondos fijos o animados. Por ejemplo: la pantalla de presentación, el logotipo pequeño en la esquina del menú o los indicadores de juego. Podemos manipular imágenes más grandes que la pantalla o de tan sólo pocos píxeles. También transformarlas de tamaño, moverlas por la pantalla e incluso detectar colisiones entre ellas. En Blitz3D disponemos de numerosas funciones para realizar todas estas operaciones. Aunque en este apartado sólo veremos las más usadas, son suficientes para cualquier proyecto de videojuego.

FUNCIONES DE DIBUJO

Debemos saber que, al igual que dibujamos en la pantalla "FrontBuffer" o en el "BakBuffer", podemos hacerlo directamente sobre una imagen con la función "ImageBuffer".

Es decir, podemos asignar que una imagen creada o cargada desde disco sea el búfer donde se dibuje. De esta forma podemos cargar una imagen y hacerle modificaciones sin que puedan verse o dibujar sobre una imagen nueva en tiempo real. En el programa "ejemplo6_5.bb" podemos ver un ejemplo de este procedimiento. (Para probar este ejemplo es necesario copiar al disco duro "C:\" el archivo de imagen "imagen.bmp" contenida en el CD). En el programa "ejemplo6_1.bb" podemos ver un ejemplo de este procedimiento.

Para cargar una de nuestras imágenes guardadas en el disco debemos utilizar la instrucción "LoadImage". Podremos utilizarlas en multitud de formatos como: bmp, jpg , gif, pcx o png. El archivo cargado lo asignamos a una variable global, la cual contendrá la dirección de memoria donde está la imagen guardada después de cargarla.

Hay varias instrucciones para dibujar nuestra imagen en la pantalla. Podemos dibujarla completa o sólo una porción rectangular de ella.

Para dibujar la imagen completa utilizando la máscara de color se utiliza "DrawImage" y para dibujarla sin ningún color transparente, "DrawBlock":

```
Global imagen
; La variable que manipula la imagen
Graphics 800,600,16
imagen=LoadImage("foto.bmp")
; Cargamos la imagen

DrawImage imagen,10,10
; Dibujamos la imagen en las
coordenadas X=10 e Y=10 de la
pantalla

DrawBlock imagen,200,10
```

Para dibujar una porción rectangular de la imagen utilizando la máscara de color se utiliza "DrawImageRect" y para dibujarla sin ningún color transparente, "DrawBlockRect".

Los parámetros para ambas son los mismos:

```
DrawImageRect variable que
manipula la imagen, posición
X, posición Y,
posición X del trozo rectangular,
posición Y del trozo rectangular,
tamaño horizontal del trozo,
tamaño vertical del trozo
```

Podemos salvar una imagen al disco en formato .bmp con la instrucción "SaveImage":

```
img = SaveImage (variable que
manipula la imagen, "fondo.bmp")
```

Si por el contrario, queremos liberar de la memoria una imagen que ya no necesitamos, utilizaremos

```
"FreeImage" : FreeImage imagen
```

TRANSFORMACIÓN DE IMÁGENES

A cualquier imagen que hayamos creado o cargado desde disco, se le puede cambiar el tamaño. Disponemos de dos funciones para esta operación "ScaleImage" y "ResizeImage ":

```
ScaleImage variable que
contiene la imagen, escala
horizontal, escala vertical
```

Con esta función escalamos la imagen por porcentajes, es decir, un valor en "escala horizontal" y "escala vertical"de 1.0 significa un tamaño del 100%. Un valor de 2,0 sería 200% y un valor 0.5 sería la mitad del tamaño 50%. Tenemos una imagen de 200x100 píxeles y queremos aumentarla el doble horizontalmente y a la mitad de altura:

```
ScaleImage imagen,
2.0,0.5
```

NOTA

Para Blitz3D, el color negro *(rgb(0,0,0))* de las imágenes es considerado como transparente y no se dibujará. A esto se le denomina *máscara de color*. Si queremos cambiar la máscara de color estándar a otro color que no sea el negro, es decir, cambiar el color transparente, debemos utilizar "MaskImage" :

```
MaskImage variable que
manipula la imagen, valor de
rojo, valor de verde, valor
de azul.
```

NOTA

Hay que tener en cuenta que el modo gráfico se tiene que definir antes de cualquier operación de carga de gráficos, si no éstos se borrarían de la memoria.

```
ResizeImage variable que
contiene la imagen, tamaño
horizontal, tamaño vertical
```

Por el contrario, con esta función aumentaremos o disminuiremos el tamaño horizontal o vertical de la imagen por cantidad de píxel. Es decir, tenemos una imagen de 300 X 200 píxeles y queremos aumentarla al doble de su tamaño:

```
ResizeImage imagen,600,400
```

También podemos rotar una imagen cualquiera con la función " RotateImage ":

```
RotateImage variable que
contiene la imagen, valor
flotante en grados desde
0.00 hasta 360.00
```

Esta función es demasiado lenta para utilizarla en tiempo real, así que lo mejor sería realizar la rotación en el búfer oculto y luego mostrarla en la pantalla. Blitz3D aplica automáticamente un proceso llamado *filtro bilinear* , el cual suaviza la imagen para evitar los dientes de sierra. Para anular la aplicación de este filtro podemoss utilizar la función " TFormFilter ":

```
TFormFilter True (1. Activado)/
False (0. Desactivado)
```

COLISIONES ENTRE IMÁGENES

Es necesario disponer de procedimientos para el manejo de sprites en el desarrollo de juegos 2D o para implementar otros elementos presentes en todo tipo de juegos como la activación de botones en los menús por medio del ratón. Nos referimos a los sistemas de colisiones entre imágenes.

Encontramos funciones extremadamente potentes para este fin: "ImagesCollide", "ImagesOverlap".

Estas funciones detectan dos imágenes a nivel de píxel ignorándose el color transparente. Para que funcione necesitamos decirle a la función las dos imágenes que queremos detectar y la posición en la pantalla de cada una de ellas. Las dos funcionan exactamente igual, a diferencia de que "ImagesOverlap" es más rápida porque sólo detecta si una imagen esta sobre la otra:

```
ImageOverlap
ImagesCollide (imagen1, posicion1
X, posicion1 Y, frames1, imagen2,
posicon2 X, posicon2 Y, frames2)
```

Frame1 y frame2 se refieren al fotograma de animación de la imagen, en caso de que tuviese. Por ejemplo, el sprite animado de tu personaje o el puntero del ratón con diferentes aspectos.

Puede ocurrir que necesitemos detectar sólo una porción de la imagen. Es muy común este hecho en las aventuras gráficas, donde el puntero del ratón tiene que interactuar con partes de la imagen del escenario. Para realizar esta operación podemos utilizar las funciones "ImagesRectCollide" y "ImagesRectOverlap". Su funcionamiento es igual que las anteriores instrucciones, a diferencia de que detecta una imagen en contacto con una porción de imagen.:

```
ImagesRectOverlap
ImagesRectCollide (imagen, posicionX
imagen, posicionY imagen,
posicionX de la porción,
posicionY de la porción,
tamaño horizontal de la porción,
tamaño vertical de la porción)
```

Disponemos además, de una función para detectar si dos porciones de la pantalla están en contacto. Se trata de " RectsOverlap ":

```
RectsOverlap ( posicionX
porcion1, posicionY porcion1,
tamaño horizontal porción 1,
tamaño vertical porción 1,
posicionX porcion 2, posicionY
porcion 2,
tamaño horizontal porción 2,
tamaño vertical porción 2)
```

Se puede observar el funcionamiento de estas funciones en los ejemplos "ejemplo6_2.bb" y "ejemplo6_3.bb" que encontraremos en el cd.

INFORMACIÓN DE IMÁGENES

Conocer el tamaño o la posición de una imagen es fundamental para implementar correctamente todas las funciones de transformación y colisión. Para ello, Blitz3D ofrece diferentes funciones que nos proporcionan esta información:
"ImageWidth",
"ImageHeight",
"ImageXHandle",
"ImageYHandle":

```
ImageWidth (variable que
contiene la imagen)
```

Proporciona el tamaño horizontal de la imagen en píxeles

```
ImageHeight (variable que
contiene la imagen)
```

Proporciona el tamaño vertical de la imagen en píxeles.

```
ImageXHandle (variable que
contiene la imagen)
```

Proporciona la posición horizontal (X) de la imagen.

```
ImageYHandle (variable que
contiene la imagen)
```

Proporciona la posición vertical (Y) de la imagen.

En el próximo número...

... estudiaremos las posibilidades 2D del Blitz3D. Conoceremos estas capacidades en un mundo 3D, empezando por el manejo de sprites.

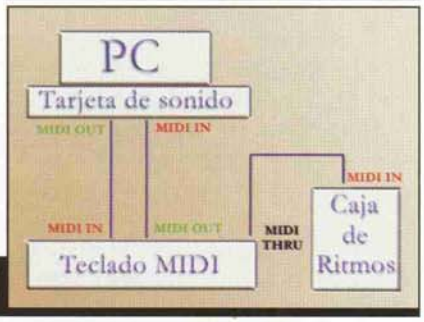
Secuenciadores. Cubase VST (III)

En esta entrega, terminaremos la serie básica de tutoriales que hemos dedicado a Cubase. Vamos a conocer el procedimiento a seguir para grabar música a través de la conexión MIDI, así como su correcta implementación en el conjunto de las pistas.

Para este tutorial, asumimos que hay un teclado MIDI conectado al ordenador por medio de la tarjeta de sonido como se muestra en la figura 1.

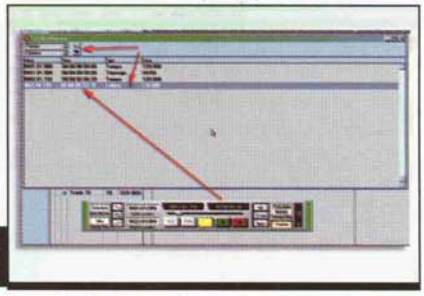
PREPARANDO LA GRABACIÓN MIDI

Antes de comenzar a grabar debemos ajustar las unidades de tiempo y ritmo en la barra de transporte o en la opción "Menu / Mastertrack / List", como se muestra en la figura 2.



1

Un esquema básico de conexión MIDI.



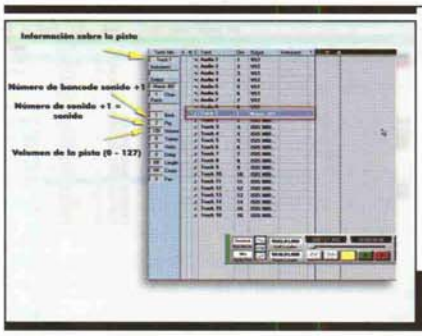
2

Procedimiento para insertar un nuevo tiempo y compás.

En la lista del Mastertrack podemos sumar todos los cambios de tiempo y de compás que queramos con solo pulsar el botón "In" después de seleccionar el punto de la canción en la barra de transporte. Aconsejamos un tiempo de 120 y un ritmo 4/4. Activamos el clic del metrónomo. Elegimos la pista donde irá la grabación. Por ejemplo, elegimos el "track 1", le asignamos un nombre (nos aseguramos de que en la casilla "C" tenemos seleccionado un tipo de pista MIDI). A continuación, debemos seleccionar el canal y salida MIDI para la pista. En la casilla "Chn" seleccionaremos el canal MIDI 1 (Disponemos de hasta 16 canales diferentes), que será el mismo que tenga el teclado maestro MIDI. En la casilla "Output" debemos seleccionar el dispositivo de salida MIDI: si se dispone de una tarjeta compatible con Sound Blaster elegiremos "SB... (MIDI Out) ...", si es otro modelo, su salida MIDI. Si por el contrario vamos a utilizar una fuente externa de sonido, determinaremos la salida de este módulo que posiblemente llevará instalado una interfaz MPU, y pondrá seguramente "Maestro MPU-401" (Fig. 3).

Para terminar nuestra preparación, debemos seleccionar el sonido y ajustar el volumen de la pista. Esta operación la podemos realizar de dos formas diferentes. Por un lado, podemos seleccionar valores en el apartado "Track Info" de la ventana de arreglo (ver figura 3).

Colocamos un "1" en la casilla "bank" (esta casilla se refiere a los bancos de sonidos, generalmente dos: 1 y 2). En La casilla "prg" (programa), colocaremos el número del sonido que queramos. La lista de sonidos empieza en el 2,



3

Descripción de "Track info" de la ventana de arreglo.

así que un sonido de piano que en la tabla corresponde al primero (1), tendríamos que activarlo colocando en esta casilla un "2". Para el sonido segundo, colocaríamos un "3" y así sucesivamente. Luego, asignaremos un volumen para esa pista en la casilla "Volume". El volumen máximo es el 127 y el mínimo el 0.

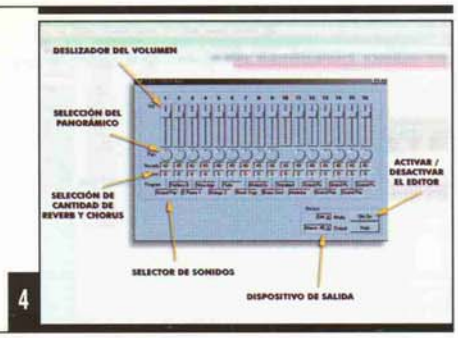
Otra forma de ajustar estos valores es mediante el Editor GM/GS/XG.

EDITOR MIDI

Para desplegar el editor GM/GS/XG pulsamos CTRL + Y o en "GM/GS/XG Editor" en el menú "Edit". Nos encontraremos ante una especie de mesa de mezclas de 16 pistas (una para cada canal MIDI). Antes de nada, en "Device", situado en la esquina inferior derecha, debemos seleccionar el dispositivo de salida como hicimos anteriormente y el modo de reproducción: GM (General MIDI), GS (Estándar de

NOTA

Si usamos una fuente de sonido externa como un módulo de sonido, debemos asegurarnos de que es compatible con General MIDI.



4

Descripción del editor MIDI GM/GS/XG.

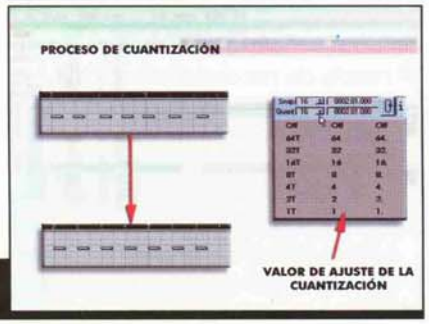
Roland) y XG (Estándar de Yamaha). Seleccionamos GM o el correspondiente al instrumento.

Para elegir el instrumento, pulsamos con el ratón en la casilla de valores de "program" de la primera pista (canal MIDI 1). En "Pan" ajustamos el panorámico y en el deslizador "Vol" el volumen de la pista. Además, podemos activar un efecto de reverberación y de *chorus* (Fig. 4).

GRABAR

Podemos seleccionar el tipo de clic del metrónomo en "Options / Metronome" entre el "beep" del ordenador o un sonido del dispositivo externo. También aquí se puede cambiar la cuenta atrás en compases antes de empezar a grabar en la casilla "precount" en número de compases "bars".

Coloquemos el localizador izquierdo de la ventana de arreglo en el comienzo de la grabación y si no sabemos hasta dónde grabaremos, desplazamos el localizador derecho hacia delante muy lejos para no quedarnos cortos. Pulsamos el botón de grabación o la tecla "*" del teclado numérico y ¡a tocar!



5

Proceso del ajuste de valores en la cuantización.

MÉTODOS DE GRABACIÓN

Podemos reemplazar (Replace) o mezclar (Overdub) las grabaciones en una misma pista activando o desactivando la casilla "Rec Mode" en la barra de transporte.

GRABANDO OTROS EVENTOS MIDI

Podemos también grabar cualquier otro elemento MIDI que no sea una nota, como por ejemplo, el pedal de sustain o la rueda de inflexión de tono al mismo tiempo que tocamos o de forma independiente. Para esto último, sólo tenemos que activar el modo de grabación "mezclar" (Overdub) y grabar. Oiremos la música mientras añadimos los nuevos eventos MIDI.

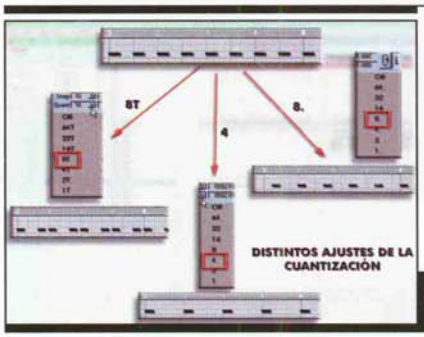
Podemos activar un ciclo de grabación y grabar en un espacio determinado de una forma continua, sumando notas a las ya existentes.

CUANTIZACIÓN

Es prácticamente imposible que la ejecución humana llegue a coincidir las notas en la medida exacta del compás. Por ello, quizás el procedimiento más importante a la hora de editar una grabación sea aplicar la cuantización.

Sin embargo, puede que no queramos ajustar nuestra grabación para que la reproducción no suene tan mecánica. Pero cuando utilicemos ritmos o un gran volumen de pistas simultáneas es imprescindible aplicar este procedimiento (Fig. 5).

Los valores de cuantización son los que utilizamos para indicarle a Cubase las posiciones exactas hacia donde mover las notas. Por ejemplo, un valor de 4 indicará una cuantización de corcheas. Hay varios métodos de cuantización que podemos activar en el menú "Functions": sobre cuantizar (Over Quantize), cuantización interactiva (Interactive Quantize), cuantización automática (Automatic Quantize), cuantización por ranuras (Groove Quantize) y cuantización analítica (Analytic Quantize). Lo más eficaz es utilizar una combinación de varias de ellas. Por



6

Diferentes ajustes de cuantización.

ejemplo, empezar realizando una "sobre cuantización" y si el resultado no es el óptimo aplicar posteriormente un "cuantización interactiva". En todo caso, el resultado debe ajustarse sin falta a nuestras necesidades. Hay que probar siempre con diferentes medidas de cuantización. Hay que tener en cuenta que esta función actúa siempre sobre las notas seleccionadas. En caso de que no haya ninguna, afectará a toda la pista (Fig. 6).

Una vez realizada la cuantización podemos congelarla con la opción "Functions / Freeze Quantize" para poder aplicar otra cuantización a la existente. También podemos volver a los valores iniciales antes de la cuantización en cualquier momento (incluso después de la grabación) con la opción "Functions / Undo Quantize".

Básicamente, ya estamos preparados para grabar y editar nuestro tema en este secuenciador. Sus posibilidades son enormes y sólo con la investigación y el uso se puede sacar el mayor rendimiento posible a esta herramienta.

DEFINICIÓN

▸ CUANTIZAR

Cuantizar no es más que mover las notas grabadas a posiciones exactas dentro del compás.

En el próximo número...

... conoceremos el que quizás sea el mejor editor de audio del mercado, el Sound Forge.

Desde los primeros juegos para PC a los arcades

Los arcades de plataformas han sido los preferidos en las recreativas de siempre, consolas y en los primeros ordenadores domésticos. El motivo principal de este éxito fue, sin duda, la riqueza visual, la incesante acción que proporcionaban y los fantásticos mundos que rodeaban a nuestro personaje.

Un juego de plataforma se caracteriza, principalmente, porque el decorado y los personajes se representan en una vista lateral. Generalmente, los escenarios donde transcurre la acción están constituidos por un fondo y decorados formados por: objetos de adorno, el suelo, el techo y las paredes. Son estos últimos elementos los que utilizan los personajes del juego, como plataformas para andar, correr o saltar.

En este número hablaremos un poco de lo que fueron y son en la actualidad estos juegos en PC, qué técnicas utilizaban y cómo evolucionaron.

LOS PRIMEROS JUEGOS EN 16 BITS PARA EL PC

Los primeros juegos que podíamos ver en un PC eran versiones de juegos existentes en sistemas de 8 bits como Spectrum 128 o MSX. Se disponía de 16 colores con tarjetas de vídeo EGA y, aunque los juegos eran exactamente iguales, carecían de un mejor atractivo visual. Prácticamente fue la propia Microsoft la que fomentó el uso del PC (clónicos del sistema IBM PC) para los desarrolladores con su simulador de vuelo *Flight Simulator*.

Pronto se empezaron a vender los compatibles PC en tiendas de electrodomésticos y aparecieron juegos exclusivos que abrieron el camino al futuro lúdico de este sistema.

Por aquel entonces, se establecieron en los PC las aventuras gráficas, sobre todo con la aparición de las tarjetas de vídeo VGA que permitían hasta 256 colores en pantalla y resoluciones más que aceptables. Cabe resaltar: las creadas por la empresa Lucas Arts como *Maniac mansión, Indiana Jones* o el clásico *Monkey Island* (1990); las producciones de Sierra, como la serie *Gabriel Knight* o el legendario *King Quest* (1984) de Roberta Williams; y la fantástica aventura *Alone in the dark* de Infogrames (1992), que introdujo la técnica de mezclar elementos 3D sobre escenarios 2D prerenderizados. Destaca también el triunfo de la Aventura Gráfica en España de la mano de Opera Soft con *La abadía del crimen*. (1989)

Gracias a los sistemas de almacenamiento que tenían los PC, estos juegos disponían de guiones extensos e infinidad de

¿SABÍAS QUE...?

La serie Comander Keen, (primeros juegos de ID Software en 1990) nació de la mano de John Carmack, quien pretendía emular la suavidad del scroll de pantalla del Mario Bros para Nintendo en el monitor del PC. Sin embargo, se encontró con el problema de la velocidad en las tarjetas de entonces y tuvo que redescubrir un sistema de impresión de tiles nuevo, creando su propio motor de scroll lateral para PC.

1

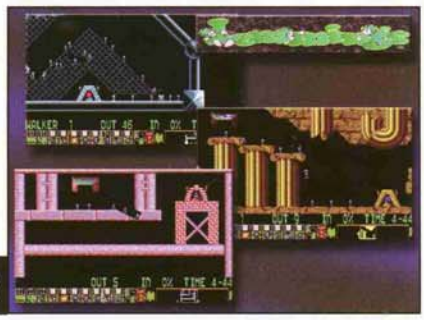
Another World y Flashback fueron los juegos de plataforma más laureados.

2

ID Software nació con la serie Commader Keen. Paralelamente, la veterana Opera Soft española triunfaba con Livingstone Supongo.

3

La serie Sims y Civilitation fueron los juegos de estrategia bandera a comienzos del PC.



4 *El más auténtico y original juego de la historia, el clásico Lemmings.*

5 *Algunas versiones de recreativas tuvieron gran éxito para PC.*

6 *Con Duke Nukem (Apogee, 1991) empezaban las aventuras del "duque".*

7 *El primer shoot´em up 3D de la historia, HoverTank. Pero fue con Wolfenstein 3D cuando realmente se consagró ID Software.*

gráficos. Sin embargo, los que seguían triunfando eran los de plataformas, eternos y fantásticos. Para PC se realizaron verdaderas joyas como la versión de *Prince of Persia* (1990), precursor del realismo en el movimiento de los personajes.

Para los primeros 4 colores (CGA) encontramos de nuevo a Opera Soft con *Livingstone Supongo* y, ya con 16 colores (EGA) aparecieron juegos de talla como la serie *Risk Dangerous* (*Core* 1989). Y no podemos dejar de hablar de los que quizás sean de los mejores juegos para PC de la primera época: nos referimos a *Another World* y su continuación *FlashBack* de Delphine Software. El primero de ellos (1991) destaca por su increíble ambientación, creada por un sistema gráfico único en su época, en el cual se disimulaba la poca cantidad de colores en pantalla disponibles por entonces con técnicas de dibujo animado muy estilizados. La continuación, Flashback (1992), invadió los comentarios de todo el mundo: unos gráficos fascinantes llenos de detalle, unos movimientos realistas y una jugabilidad cargada de estrategia. Recordaba mucho al *Prince of Persia* y quizás sea el más versionado para otras plataformas.

Un género afloró con más fuerza en la época del PC: la estrategia. Empezamos con el innovador juego de ajedrez *Battle Chess* (1989) de InterPlay, en donde las piezas estaban dotadas de movimiento y se peleaban entre sí. Pero la estrategia pura nació de la mano de Maxis con la saga *Sims*: *Sim City* (1989), *Sim Earth* (1990), *Sim Farm* (1993). Del señor Sid Meier nace uno de los juegos de estrategia por turnos más famosos, *Civilization* (Microprose, 1991), donde controlas toda la historia de la humanidad. Sin embargo, en esta época apareció en el mercado el juego más original de la historia: *Lemmigns* (*Psygnosis*, 1991). En este juego debes llevar a un grupo de pequeños seres hacia la puerta de salida de cada nivel. Poseía unos movimientos suavísimos y perfectos, lo que permitía distinguir perfectamente todas las acciones a pesar del reducido tamaño de los personajes.

En general, entre los años 84 y 94 el género que más abundaba era el arcade y aparecieron infinidad de títulos con calidad suficiente para convertirse en clásicos. Versiones de recreativas como el *Pang, Pac Man, Tetris, Arkanoid* (1987), *Double Dragon, Paper Boy* y el fantástico *Golden Axe*. Apareció por entonces también una conversión para PC de la famosa serie *Dragon´s Lair* que introdujo un nuevo sistema de jugabilidad.

NOTA

No fue, en realidad, Wolfenstein 3D el primer shoot´em up 3D que existió, aunque sí el más conocido mundialmente. Anteriormente a la publicación de este juego, ID Software publicó en Abril 1991 HoverTank 3D. Fue el primer juego 3D que realizó John Carmack. Después, y junto a John Romero (quién inventó la técnica del mapeado de texturas), realizaron Catacombs 3D (finales de 1991), donde aplicaron su nueva técnica, añadiendo al juego paredes de ladrillos y hierba al suelo. No se quedaron ahí, ya que quisieron que el jugador participase más en el juego y colocaron delante de la cámara un brazo con un arma. Lo demás ya es historia.

En el próximo número...

... empezaremos a comentar cada uno de los géneros de juegos para PC individualmente. Hablaremos de los juegos más significativos de los arcades de plataformas y de su técnica.

Cuestionario Videojuegos 6

Preguntas

1. En la frase "Mi coche es verde", ¿cómo puedo eliminar la palabra "es" y quedarnos con la frase "Mi coche verde" en Blitz3D?
2. Describe la instrucción que nos permite detectar, por ejemplo, el cursor del ratón con un botón del menú del juego.
3. Señala las partes que componen la presentación de un juego, en qué orden van y cuál de ellas es estrictamente necesaria.
4. ¿En qué partes se puede dividir el menú principal de un juego?
5. ¿Cómo se puede crear más o menos espacio entre las letras que componen una fuente en el programa Font Creator?
6. ¿Qué formato de objeto utilizaremos para las edificaciones y cuál para la bionave de combate?
7. Hemos grabado un sonido con el GoldWave en nuestro ordenador. Al reproducirlo oímos que suena demasiado bajo. ¿Cómo podemos aumentar el sonido al máximo manteniendo su calidad?
8. ¿Cómo podemos crear con GoldWave un sonido propio sin obtenerlo del exterior?
9. Queremos seleccionar un sonido y aumentar el volumen de la pista en CUBASE. ¿De qué dos formas diferentes podemos realizar esta operación?
10. Queremos ajustar las notas de una grabación a las posiciones exactas dentro de un compás en CUBASE. ¿Qué técnica debemos utilizar?

Respuestas al cuestionario 5

▷ **1.**
```
Type nave_enemiga
   Field x,y,z
   Field estado
   Field grafico
End Type
```

▷ **2.**
```
If Windowed3D() and GfxMode3DExists(800,600,16)
   Graphics3D 800,600,2
Endif
```

▷ **3.** Colocando el efecto de niebla con un rango de visión anterior al de la cámara.

▷ **4.** Utilizando el sistema de tileado, consistente en repetir la textura hasta cubrir todo el terreno.

▷ **5.** Seleccionando la herramienta "Object Selector" y pulsando sobre el borde de la figura.

▷ **6.** Rasterizar una capa significa convertirla a mapa de bits. Se utiliza para poder aplicar efectos de imagen a una capa de vectores.

▷ **7.** El nuevo dispositivo para reproducir o grabar se elige del panel "Device Controls Properties" en la opción "Device".

▷ **8.** Utilizando marcas con "Cue Points" del menú "Tools".

▷ **9.** Pulsando CRTL + T.

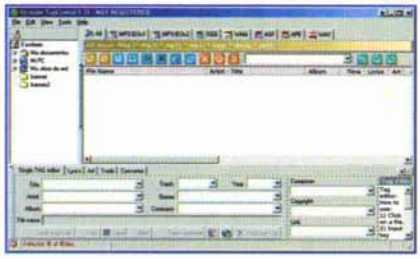
▷ **10.** Con la herramienta de línea, por ejemplo en el editor de teclas.

Contenido CD-ROM 6

▶ AUDIO

■ Airmix Sound Automation 9.5
Podrás mezclar música en variados formatos de audio, todo en tiempo real.

■ Abander Tag Control 1.3



Gracias a esta útil herramienta podrás manejar fácilmente toda tu colección de ficheros de audio.

■ DRS 2006. The Radio Automation Software 2.1
Potentísimo programa con el que podremos convertir nuestro PC en una auténtica máquina musical.

■ Expresion Tone Generator 2.0
Divertida aplicación con la cual podrás crear interesantes efectos de sonido a partir de expresiones algebraicas.

■ Mistergenerator 1.0
Mezcla tus archivos de audio en formato .wav fácilmente.

▶ DISEÑO 2D

■ Font Creator 3.0
Con esta herramienta podrás crear y modificar fuentes True Type, para realizar la tipografía del juego.

■ Evil Foto 3
Completo y potente programa de catalogación de imágenes que sin duda nos será muy útil.

■ MagicImageMarker 1.3
Ponle marcas de agua a tus imágenes, ya sea individualmente o en grupo.

■ Pics Print 1.16
Imprime tus imágenes digitales fácilmente y con una gran calidad.

■ Professor Franklins Instant Photo Effects 1.1
Crea rápidamente sorprendentes efectos en tus imágenes.

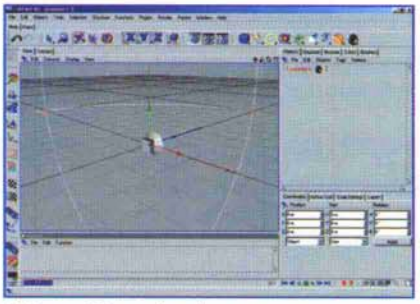
▶ DISEÑO 3D

■ Famous 3D
Simpático programa de animación en 3D que además podremos integrar en nuestros contactos de MSN.

■ Pixel 3D 1.1
Produce gráficos 3D de calidad a partir de imágenes en dos dimensiones.

■ Cinema 4D XL7
Gracias a este potente programa, podremos renderizar, animar personajes y crear avanzados efectos de partículas.

■ Caligari True Space 5



Anima tridimensionalmente con esta potente herramienta.

▶ PROGRAMACIÓN

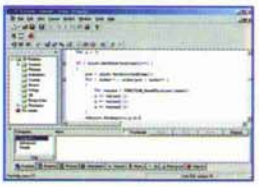
■ 3D Rad
Entorno de desarrollo para que creemos nuestros propios juegos en tres dimensiones.

■ DLL to Lib 1.4
Convierte una DLL en su equivalente librería estática gracias a esta herramienta.

■ LibCON C++ Game Programming Library 0.72 b
API para C++ que te ayudará en la programación de tus juegos.

■ Multimedia Fusion 1.5
Crea juegos y otras completas aplicaciones multimedia usando este programa.

■ Jamagic 1.2
Completo entorno de desarrollo para crear juegos en 3D.



▶ JUEGOS

■ Prince of Persia
El primer juego en el que los movimientos de los personajes estaban dotados de un gran realismo.

■ Lost Vikings

Todo un clásico que, aun así, sigue siendo divertidísimo. Un buen juego, en definitiva.

■ Pandemonium
Uno de los primeros juegos en tres dimensiones que obtuvo muy buenos resultados para la época.

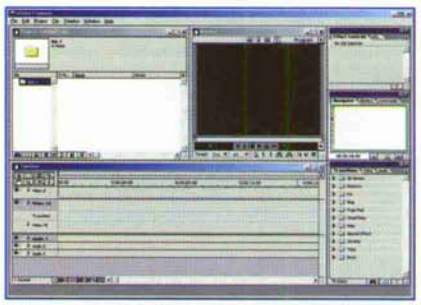
■ Rayman 2
Maravilloso juego, adaptación del primer título (Rayman) en 3D.

■ Zone of Fighters
Nueva versión de nuestro juego.

▶ VÍDEO

■ VC Video Library 1.1
Ten bien ordenada y clasificada tu colección de vídeos gracias a este catalogador.

■ Adobe Premiere 6.0



Demo del excelente programa de Adobe, uno de los más utilizados para editar vídeo.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.

Construcción de herramientas propias para el desarrollo del juego

En este número haremos un inciso antes de comenzar a programar el juego. Generalmente es muy común el desarrollo de otro tipo de aplicaciones que ayuden a los grafistas y programadores en el proceso de construcción de un juego. Nos referimos a pequeños programas que sirvan para chequear gráficos.

En nuestro caso, vamos a desarrollar un visualizador de terrenos y otro de modelos. También realizaremos una pieza importante en el conjunto de juegos cuyos mapeados se basan en zonas diferentes: los editores de niveles. Ya que en "Zone of Fighters" la acción transcurre en zonas de combate, desarrollaremos un editor para crearlas a gusto del jugador. Esta herramienta ayudará a alargar la vida del juego y añadirá un atractivo más para el consumidor.

VISUALIZADOR DE TERRENOS

Con esta aplicación, podremos generar un terreno a partir de un mapa de alturas, aplicarle texturas y recorrerlo con la cámara. Básicamente, así será nuestro visualizador de terrenos. Sin embargo, habrá elementos en este número que no hemos visto aún, aunque esto no significa ningún problema; lo importante es que se entienda la estructura básica de esta aplicación. De todas formas, todo el código contenido en el fichero "vis_terreno.bb" del CD está comentado y lo que no se entienda ahora se comprenderá perfectamente más adelante.

El núcleo de este visualizador, y del de modelos, nos servirá para realizar el editor de zonas de combate. En el CD-ROM puedes ver una imagen con la estructura del programa en un sencillo pseudocódigo.

Observamos en esta estructura que el programa está dividido en dos partes. Una primera que constituye el núcleo principal encerrado en un bucle, y una segunda parte compuesta por las distintas funciones o tareas que se realizan. Vamos a explicar los procesos más significativos del código. La mayoría de las instrucciones las hemos visto en la sección de Blitz3D. Es conveniente que tengas el código a mano mientras lees las explicaciones.

Antes de entrar en el bucle principal que gobierna el programa, debemos definir el modo gráfico y activar el doble búfer:

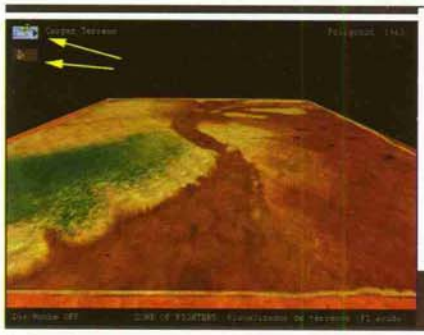
```
Graphics3D 640,480,16,1
SetBuffer BackBuffer()
```

Prepararemos el entorno donde se colocará el terreno y definiremos todas las constantes, variables y tipos utilizados.

La interfaz de usuario constará de dos iconos en la parte superior de la pantalla que servirán para cargar un terreno o salir del programa (Fig. 1).

Para ello precisaremos de un tipo que defina el gráfico, coordenadas y un literal con el nombre de la acción que representa. Aunque sólo tenemos dos, esta estructura la usaremos luego para el editor de zonas donde tendremos muchos más.

```
Type botones
      Field grafico
      Field x,y
      Field nombre$
```



1

La interfaz de usuario del visualizador de terrenos está compuesta por dos iconos en la parte superior de la pantalla.

```
End Type
Dim boton.botones(1)
Dim botonB.botones(1)
```

La matriz nos servirá para añadir más botones si lo necesitáramos más adelante. Definimos *botonB* porque será el icono de otro color que se mostrará cuando estemos encima con el cursor. A continuación, cargamos la imagen del cursor y definimos el resto de variables globales utilizadas en el programa. Antes de entrar en el bucle principal, colocamos el cursor del ratón en el centro de la pantalla *(MoveMouse 320,240)* y llamamos a la función para crear las luces y la cámara (luego explicaremos cada una de las funciones detenidamente). En la línea:

```
Collisions ENTIDAD_CAMARA,
ENTIDAD_TERRENO,2,3
```

Llamamos a la función de colisión del Blitz3D para que la cámara detecte al terreno y se deslice por él. Empezamos el bucle principal borrando el búfer actual de dibujo (la pantalla) con *Cls*, seguidamente detectamos desde el teclado las opciones: salir, activar o desactivar la niebla, activar el

paso del día a la noche y convertir el terreno en modo wireframe (en alambre, sin texturas). El sistema de *Switch* nos ayudará a conmutar valores para activar o desactivar acciones, simplemente se trata de definir variables cuyo valor será 0 ó 1.

Por ejemplo, en la línea:

```
If KeyDown(4)
SW_DiaNoche=1-SW_DiaNoche
```

Si pulsamos la tecla "3" el valor de SW_DiaNoche valdrá 1 menos su antiguo valor:

Si SW_DiaNoche=0

```
SW_DiaNoche=1 - 0 ‡ 1
```

Si por el contrario vale 1

```
SW_DiaNoche=1 - 1 ‡ 0
```

De esta forma, con una sola instrucción podemos asignar un 0 o un 1 al Switch SW_DiaNoche

Para controlar la rotación de la cámara utilizaremos el ratón; así que, mientras pulsemos el botón derecho podremos moverla.

Para girar la cámara mediante el ratón utilizamos una serie de variables que almacenarán la velocidad de movimiento horizontal (eje X) o vertical (eje Y) de éste.

En mx y my almacenamos la velocidad relativa y la convertimos a otra absoluta:

```
mx#=mx#-Float(MouseYSpeed())/
-7 : mx=mx/1.5
my#=my#+Float(MouseXSpeed())/
-7 : my=my/1.5
```

Esto nos servirá para sumar grados al giro en las variables X e Y

```
X#=EntityPitch(camara)+mx
```

Giro en vertical (EntityPitch) + los grados del movimiento del ratón (mx)

```
Y#=EntityYaw(camara)+my
```

Giro en horizontal (EntityYaw) + los grados del movimiento del ratón (my)

A continuación, limitamos el giro vertical a sólo 180 grados en los dos sentidos

```
If X>=89 And X<=180 X=89
If X<=-89 And X>=-180 X=-89
```

Seguidamente, hacemos efectivos los giros de ambos ejes y volvemos a colocar el cursor del ratón en la mitad de la pantalla para no perder el desplazamiento continuo de éste.

Después de actualizar y dibujar el mundo 3D y antes de intercambiar los búferes, debemos dibujar y detectar la colisión del cursor con los iconos, así como la impresión de los textos de información y ayuda en pantalla.

```
For n=0 To 1
  DrawImage boton(n)\grafico,
  boton(n)\x,boton(n)\y
  If ImagesOverlap(cursor,
  MouseX(),MouseY(),boton(n)\
  grafico,boton(n)\x,boton(n)\y)
    DrawImage botonB(n)\grafico,
    boton(n)\x,boton(n)\y
    Text 60,boton(n)\y,
    boton(n)\nombre$
    If MouseDown(1)
    eleccion=n Else eleccion=-1
  EndIf
```

Con este bucle dibujamos los iconos y posteriormente utilizamos una sentencia "If" para detectar la colisión de éstos con el cursor. Si hay colisión, cambiamos de icono y si además pulsamos el botón izquierdo, guardamos en "elección" el número del icono detectado.

Vamos a explicar las distintas funciones que nos encontramos en el programa. En primer lugar, en la función *Crea_Entorno()* asignamos una intensidad determinada a la luz ambiental, ésta afectará a todas las luces que creemos a partir de este momento. Después, creamos el Sol, como luz direccional, con un color determinado y con una inclinación de 45 grados.

Posteriormente, debemos crear una cámara para poder ver el entorno 3D y la sumamos al mapa de colisiones. Asignamos un color de fondo y el mismo color para la niebla, y posicionamos la cámara para que apunte hacia el terreno y lo encuadre en la pantalla.

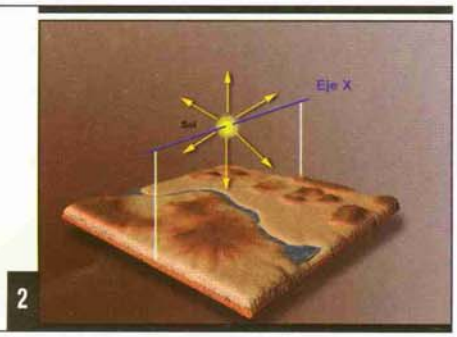
En la siguiente función *Cargar_Terreno()*, obtenemos el nombre del mapa de alturas y de las texturas del terreno para poder crearlo y asignarle el aspecto. Disponemos de tres bucles *"Repeat ... Until"*, uno para cada petición. En cada uno de ellos colocamos una instrucción *Input* que obtendrá el nombre del fichero de manos del usuario. Además, con la función *FileExists()*, nos aseguramos que dicho fichero existe antes de seguir. Continuamos, borrando el terreno que esté en pantalla; si no hiciéramos esto, al crear otro, se sumaría al anterior. Utilizamos un Switch (SW_carga) para saber si el terreno que vamos a crear es o no el primero, porque si borramos sin tener nada creado el programa dará un error. Para terminar, creamos el terreno a partir del mapa de alturas, lo sumamos a la lista de colisiones y le asignamos 2000 polígonos para su creación. Además, activamos el sistema LOD (nivel de detalle) y el sombreado.

Para finalizar, le asignamos una primera textura que lo cubrirá completamente y otra mezclada de forma tileada para proporcionar más detalle.

En la función *Salir()*, nos aseguramos de que el usuario desea salir, en caso contrario regresamos al bucle principal y si es correcto finalizamos el programa *(End)*.

La función *Dia_Noche()* es la que se encarga de disminuir y aumentar la intensidad de luz ambiental en:

```
If luz#<10 variacion_luz=
(variacion_luz)*-1
If luz#>254 variacion_luz=
```



2

Un esquema gráfico de cómo se mueve la luz del sol en el visualizador de terrenos para generar el paso del día a la noche.

```
(variacion_luz)*-1
luz#=luz-variacion_luz#
```

Las sentencias anteriores se basan en que cualquier valor positivo multiplicado por –1 se convierte en negativo y si el valor es negativo se convertirá en positivo. De este modo, realizamos estas operaciones con la variable *variacion_luz* para el cambio de intensidad de luz.

Continuamos, rotando la luz en su eje X para simular el movimiento del Sol (Ver Fig. 2).

VISUALIZADOR DE MODELOS

La estructura de esta aplicación es exactamente igual que la anterior. La única diferencia es que en vez de cargar mapas de alturas y crear terrenos, cargamos objetos 3D (Fig. 8).

Ahora esta operación la realizamos en la función *Carga_Modelo()*. Aquí, una vez cargado el modelo y su textura, detectamos de qué tipo es según la extensión del fichero.

```
If Right(modelo$,3)="3ds"
  entidad_modelo=LoadMesh(modelo$)
EndIf
...
```

En caso de ser un archivo .3DS, simplemente lo cargamos, le asignamos la textura y lo posicionamos delante de la cámara. Por el contrario, si se trata de archivos con animación (bien .B3D o .MD2) cargamos el modelo, lo animamos, le asignamos la textura y lo posicionamos delante de la cámara.

EDITOR DE ZONAS

Esta herramienta es muy útil para diseñar niveles con el mínimo esfuerzo. Básicamente, la idea es colocar, a nuestro gusto, los objetos de decorado, animales y plantas sobre un terreno determinado y grabar los datos en un fichero. Este archivo será posteriormente leído por el juego, el cual volverá a colocar todos los elementos sobre el terreno en la misma posición.

El éxito o no de nuestro editor dependerá de su facilidad de manejo. Para ello, vamos a proporcionar una vista totalmente tridimensional del terreno al usuario, el cual podrá moverse libremente por él y colocar o editar los diferentes elementos con el ratón *in situ*. De esta forma, tendrá una visión exacta de cómo se verá su nivel en el juego.

La estructura básica del programa se inspira en los dos visualizadores anteriores. En el CD-ROM podemos ver un pseudocódigo de su funcionamiento.

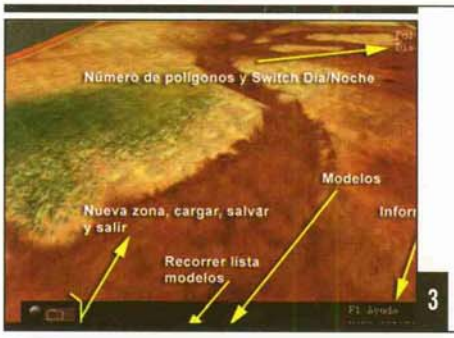
Vamos a explicar sólo las partes más importantes del código. De todas formas, en el CD-ROM se halla también el código fuente totalmente comentado ("editor.bb")

Lo fundamental del programa es que su funcionamiento se basa en el manejo de dos estructuras principales. Una primera:

```
Type tipo_mesh
      Field x#,y#,z#,ry#
      Field entidad
      Field Objeto
      Field tipo_entidad$
End Type
```

Es utilizada para manejar todos lo objetos que se están colocando y editando para crear el nivel y una segunda estructura con campos ligeramente diferentes, para crear el archivo definitivo que contendrá el nivel o zona de combate:

```
Type Tipo_archivo
      Field Tipo_entidad$
      Field entidad
      Field Identi_entidad#
      Field x#,y#,z#
      Field ry#
End Type
```



3

En la imagen se muestra una descripción de los elementos del interfaz de usuario del editor de zonas.

Para crear muchos objetos con la misma estructura disponemos de la matriz:

```
Dim mesh.tipo_mesh
(numero_maximo_mesh)
```

Esta matriz de objetos la controlamos con dos índices: *indice* e *indice_mesh*. El primero lo utilizamos en el bucle principal para pasarle a la función *Coloca_objeto()*, información acerca de qué objeto hemos seleccionado *(Item3D(indice))* y de cómo se llama.

Hemos creado una interfaz gráfica más completa con información en pantalla acerca de los objetos y la posibilidad de elegirlos con el ratón de forma visual, mediante dos iconos de flechas (Ver Fig. 3).

Una vez que hemos elegido con el ratón sobre el objeto que queremos colocar, pasamos a la función que nos permitirá realizar esta operación: *Coloca_objeto()*. A esta función llegamos con la información que nos ha proporcionado la variable *indice* sobre el objeto que vamos a colocar. El primer valor m nos servirá para asignarle el mesh (la forma) correspondiente al nuevo objeto:

```
mesh(indice_mesh)=
New tipo_mesh
mesh(indice_mesh)\entidad=
CopyEntity(m) fl
```

y el valor obj para poder identificar el nuevo objeto dándole un nombre:

```
Select obj fl
  Case 0: mesh(indice_mesh)\
tipo_entidad$="almacen1"
  Case 1: mesh(indice_mesh)\
tipo_entidad$="almacen2"
  ...
```

A continuación aplicamos la función mágica que nos permitirá mover el objeto sobre el terreno 3D mediante el ratón: CameraPick:

```
CameraPick(camara,MouseX(),
MouseY())
mesh(indice_mesh)\x#=PickedX()
mesh(indice_mesh)\z#=PickedZ()
```

PickedX() y *PickedZ()* nos proporcionan también las coordenadas exactas del terreno donde tenemos el cursor.

Como ya veremos, esta función sólo es válida si hemos asignado con anterioridad el terreno en modo "pickeable" (desmontado en polígonos) con la función *EntityPickMode*.

Después de los comandos típicos para mover el objeto con precisión mediante las teclas, detectamos si hemos hecho clic con el ratón para saber si el objeto lo colocamos definitivamente o no. En caso afirmativo, añadimos el nuevo objeto con los datos que nos interesan para su lectura posterior en el juego e incrementamos el índice global de la matriz de objetos para situarnos en uno nuevo.

```
If MouseHit(1)
  Archivo.Tipo_archivo=
  New Tipo_archivo
  Archivo\Tipo_entidad$=
  mesh(indice_mesh)\
  tipo_entidad$
  Archivo\Identi_entidad#=
  mesh(indice_mesh)\entidad
  Archivo\entidad=
  mesh(indice_mesh)\Objeto
  Archivo\x#=EntityX
  (mesh(indice_mesh)\entidad)
  Archivo\y#=EntityY
  (mesh(indice_mesh)\entidad)
  Archivo\z#=EntityZ
  (mesh(indice_mesh)\entidad)
  Archivo\ry#=mesh
  (indice_mesh)\ry#
  pick=0
  coloca=0
  indice_mesh=indice_mesh+1
EndIf
```

La función *Modificar_objeto()* parece algo complicada, pero en realidad es bien sencilla. Lo que hacemos es recorrer mediante un bucle *For ... Next* la matriz principal de objetos para buscar con cuál de ellos ha contactado el cursor en el terreno. Para aislarlo, recorremos con *For ... =Each* la estructura de cada uno de los objetos de esta matriz. Para ganar velocidad en el proceso de edición, reducimos la búsqueda sólo en los objetos que se encuentran en la vista de la cámara, con la siguiente condición:

```
If EntityInView(mesh(n)\
entidad,camara)=True
```

Además, debemos convertir el objeto en "pickeable" para poder identificarlo cuando lo detecte el cursor. Una vez identificado, se le desactiva el modo "pickeable". De esta forma se gana más velocidad en el proceso:

```
EntityPickMode mesh(n)\
entidad,3 fl Acivar
Entidad=CameraPick
(camara,MouseX(),MouseY())
EntityPickMode mesh(n)\
entidad,0 fl Desactivar
```

Para finalizar, haya cambios o no en el objeto, lo actualizamos en la estructura del archivo.

Para crear una zona nueva debemos borrar todos los objetos creados de la matriz de objetos y de la estructura del archivo de zona (función *Nueva_Zona*). Si lo que queremos es grabar la zona, simplemente damos un nombre para crear el fichero en el disco y grabar campo por campo todos los objetos contenidos en la estructura *Tipo_Archivo* (función *Grabar_Zona*). Después,

4

Dos bonitos ejemplos que muestran la posibilidad de recorrer con la cámara todos los rincones de la zona editada.

inicializamos la cámara para obtener una vista general del terreno y así poder sacar una instantánea de la zona (captura de pantalla), que utilizaremos en el juego para la elección del nivel por el usuario.

Y concluimos, con la función *Cargar_Zona()*, que simplemente lee todas las estructuras de objetos contenidas en el fichero, luego se va colocando en la matriz de objetos y, finalmente, se posiciona en el terreno.

Hemos tenido un duro trabajo en esta entrega, pero a medida que avancemos en los conocimientos sobre Blitz3D, será muy fácil adaptar estas aplicaciones a vuestras necesidades. Una buena terapia de perfeccionamiento y comprensión es hacer muchas pruebas modificando el código y así poder estudiar diferentes comportamientos en las funciones. Hay que prestar una especial atención al manejo de estructuras de datos, de las cuales se ha podido observar su importancia en las herramientas que hemos visto.

En el próximo número...

... vamos a entrar en el desarrollo del código de "Zone of Fighters". Empezaremos creando una estructura global del juego mediante pseudocódigos, importantísima para llevar un buen control en esta importante empresa.

Grafismo 3d. Modelado de la bionave de combate

Por fin ha llegado el momento de convertir en un modelo 3D el diseño de la bionave de combate. Para ello, vamos a modelar con el programa MilkShape 3D por el método de primitivas base, procurando mantener un número de polígonos razonable.

Antes de empezar vamos a organizar nuestros ficheros. Por ejemplo, creemos una carpeta principal en el disco duro llamada "juego_ZOF". Ahí crearemos las demás carpetas temporales que guardarán todo el trabajo. Al terminar, los ficheros finales resultantes se pasaran a una carpeta definitiva que usaremos en la instalación del juego. Dentro de esta carpeta creamos otra llamada "Bionave" y dentro de esta, otra llamada "Modelado".

Al final resultaría: "C:\juego_ZOF\Bionave\Modelado".

Dividiremos nuestro trabajo en partes. En la figura 1 podemos ver un esquema de las distintas piezas que forman la bionave.

CUERPO DE LA NAVE

La estructura general de la bionave es rectangular, luego se le añaden los demás elementos como las alas, alerones, cañones, etc. El cuerpo principal está dividido en dos partes, una sobre la otra. Para la parte inferior partimos de una caja a la que le damos la forma rectangular con la opción de escalado (F4). A continuación vamos a seleccionar y mover los vértices del plano inferior hacia dentro para estilizar las formas como se muestra en la figura 2.

Realizamos la misma operación pero a la inversa para la parte superior.

Una vez que tengamos las dos partes juntas, debemos unirlas físicamente por los vértices con la opción "Snap Together" (CTRL+N).

Seleccionamos los vértices de la esquina de ambas partes con la opción "Ignore Backfaces" seleccionada y unimos (Ver Fig. 3).

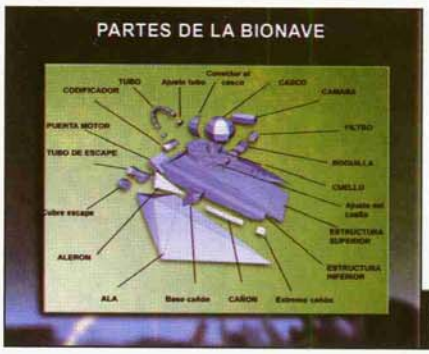
AÑADIR LAS ALAS

Una vez modelado el cuerpo principal de la bionave, iremos creando y añadiendo el resto de las piezas. Seguiremos con las alas. Éstas tienen forma de flecha y se pueden formar perfectamente a partir de un rectángulo. Una vez creado éste, seleccionamos los dos vértices exteriores traseros como se muestra en la figura 4 y lo unimos con CTRL+N.

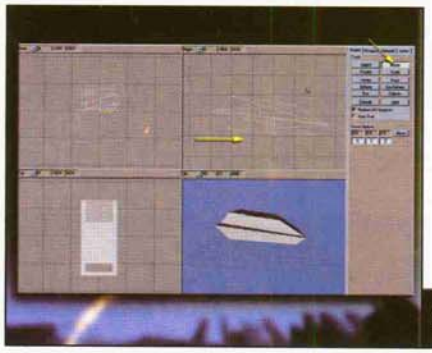
A continuación, con la opción "move" (F2), los desplazamos hacia el exterior, hasta obtener la forma deseada (Fig 4). Haremos las mismas operaciones con los demás vértices. Cuando hayamos terminado con una de las alas, la duplicamos con CTRL +D y hacemos un "Mirror Left <-> Right" (menú "Vertex") para crear y colocar la otra ala.

ESCAPE Y CUBRE ESCAPE

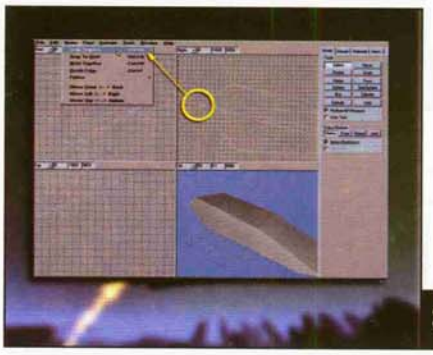
Para el tubo de escape, partimos de un cilindro de dos "Stacks" (partes) y ocho "Slices" (divisiones). Seleccionamos todos los vértices ("Ignore Backfaces" deseleccionado) de la parte central y los desplazamos



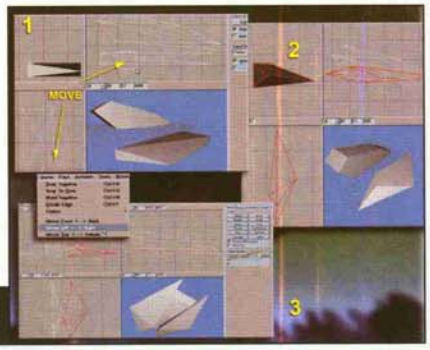
Una gráfica de las distintas partes que componen la bionave de combate.



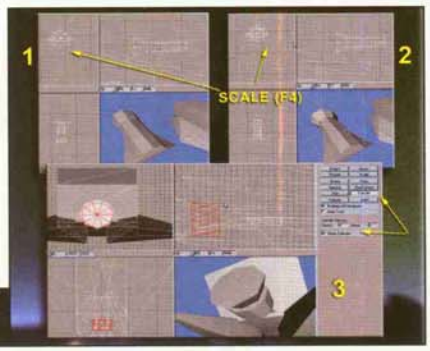
Primer proceso para crear el cuerpo de la nave.



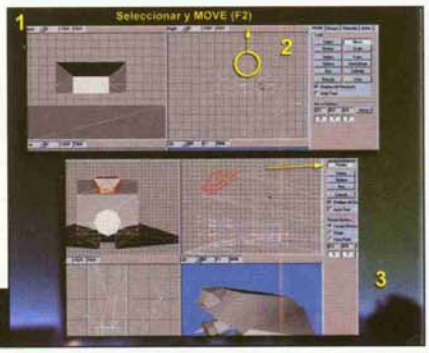
Uniendo las dos partes del cuerpo de la bionave.



A partir de un rectángulo se puede obtener las alas con sólo desplazar los vértices.



Para crear objetos como el escape a partir de cilindros, es importante escalar y mover vértices.



Un sencillo proceso para crear el codificador.

El proceso para obtener la base del cañón puede resultar algo complicado, pero lo importante es obtener una forma en "L".

hacia el lado del final del escape. La forma de éste la realizamos mediante el escalado de vértices seleccionados como se muestra en la figura 5.

Con este procedimiento de escalado obtendremos el aspecto final. Para terminar, rotamos (F3) el escape 90 grados en la vista "Right" y lo situamos en el extremo inferior trasero de la bionave. Para el cubre-escape creamos un cilindro de una parte y diez divisiones, rotamos de igual forma y lo colocamos (ver Fig. 5).

CODIFICADOR

El codificador es la pieza que traduce toda la información que viaja a través del tubo desde la cabeza del luchador hacia la bionave. Para realizarla, partiremos de un cilindro estrecho de seis divisiones en el que desplazamos hacia el exterior uno de los vértices centrales. Realizado esto, rotamos la pieza y la encajamos en la parte trasera de la bionave (ver Fig. 6).

CAÑONES

La bionave posee dos cañones situados a ambos lados. Están formados por tres piezas: un soporte, el cilindro del cañón y el extremo. El soporte tiene forma de "L" y parte de un cilindro de seis divisiones y un "stack", al que se le han desplazado los vértices como se muestra en la figura 7.

Una vez adquirida la forma se duplica (CTRL+D) y se le realiza la función "Mirror Left <-> Right". Para el cuerpo del cañón, basta con un cilindro de dos partes y tres divisiones, al cual alargamos hasta obtener la longitud adecuada. El extremo se realiza exactamente de la misma manera, aunque a partir de un cilindro con una sola parte para ahorrar polígonos. Terminamos realizando las misma operaciones de copiado y espejo para crear el otro cañón (Fig. 8).

AJUSTE DEL CASCO Y DEL CUELLO

Estas piezas situadas en la parte superior del vehículo son la base de la cabeza del protagonista. Ambas están formadas por cilindros de diez divisiones y una parte debidamente rotadas y colocadas (ver Fig. 9).

LA TAPA DEL MOTOR

Ésta es una pieza relativamente pequeña que está situada en la parte trasera sobre el tubo de escape. Partimos de una caja a la que le desplazamos los vértices como se muestra en la figura 10.

CASCO Y CONECTOR

Éstas son las piezas que engloban a la cabeza del conductor de la bionave. Están formadas a partir de sendas esferas de seis partes por seis divisiones. El conector cubre la mitad trasera del casco. Para realizarlo, debemos dividir una esfera en dos, seleccionar la mitad de sus vértices con "Ignore Backfaces" deseleccionado y borrar. Seguidamente, desplazamos la mitad restante hasta cubrir la parte trasera del casco (Fig. 11).

FILTRO DE AIRE Y CÁMARA ESPECIAL

En el juego, la cámara permite aumentar el rango de visión del casco y básicamente está modelado a partir de un cilindro achatado de cinco divisiones y una parte.

El filtro está situado en el casco y sirve para filtrar el polvo y demás elementos nocivos del viciado aire de los terrarios. Está formado por la boquilla y un filtro a cada lado. Para la boquilla creamos un cilindro de ocho partes, seleccionamos los tres vértices exteriores (cuatro caras) y los desplazamos (F2) en la

vista "Right" un poco hacia fuera. Añadimos los filtros, modelando uno de ellos a partir de un cilindro igual a la boquilla pero con uno de sus lados más anchos (seleccionar vértices y escalar en la vista frontal). Para terminar, duplicamos y realizamos un "Mirror" (Fig. 12).

TUBO Y PIEZAS DE AJUSTE

Para las piezas de ajuste del tubo al casco y al codificador, dispondremos de pequeños cilindros con ocho divisiones. Después de escalarlos y rotarlos adecuadamente, los colocamos en los lugares que se muestran en la figura 13.

El tubo de comunicación entre la nave y el piloto tiene forma curva y está formado por ocho cilindros con ocho divisiones y una parte cada uno. Es suficiente con crear uno de estos cilindros. Posteriormente, basta con duplicarlo (CTRL+D) y desplazarlo hacia arriba. La forma curva la iremos construyendo por medio de la rotación de los cilindros que vayamos duplicando. Para rotar activamos en "Rotate Options" la opción "User Point" (punto de usuario) para poder girar el cilindro a partir del puntero del ratón (Fig. 14).

Otra forma de crear el tubo sería uniendo cada cilindro entre sí por medio de los vértices de sus extremos. Es una tarea más laboriosa pero el resultado es más perfecto. El problema radica en que hay que tener todos los cilindros correctamente orientados antes de unirlos, y así conseguir la forma curva adecuada. Creamos un cilindro de ocho divisiones, lo duplicamos, rotamos y desplazamos como en el proceso anterior. A continuación, con "ignore Backfaces" deseleccionado, seleccionamos los vértices de los extremos de ambos cilindros y los unimos con "Snap Together" (CTRL+N).

Volvemos a seleccionar "Ignore Backfaces" para elegir los vértices que quedan por detrás de la vista, y unimos. Realizamos las mismas operaciones para todos los demás vértices. Para completar el ensamblaje, necesitaremos cambiar a la vista frontal para unir los vértices ocultos más difíciles.

Y por último, cambiamos a "wireframe" la vista 3D para descubrir que falta por unir los vértices centrales del cilindro.

Aunque cada cilindro esté unido, siguen siendo piezas independientes en el conjunto del tubo.

TRUCO

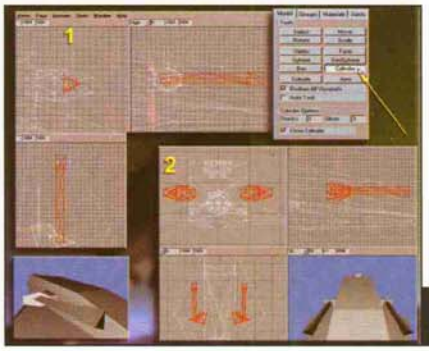
Para evitar que los cilindros no encajen bien, una buena opción sería rotar una vez realizada la duplicación y posteriormente desplazar hacia arriba como se muestra en la figura 15.

TRUCO

Si al terminar de unir todos los cilindros no se consigue la forma de la curva adecuada, una buena solución sería seleccionarlos individualmente y rotarlos hasta conseguir la curvatura. Al estar soldados, se desplazará también la unión.

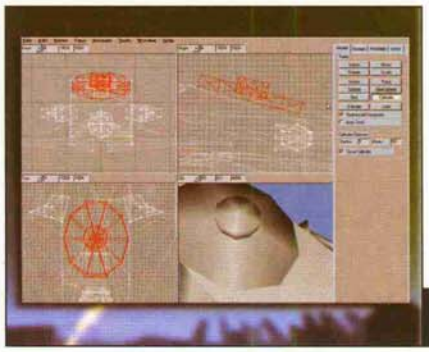
AGRUPANDO PIEZAS

Milkshape3D permite la opción de nombrar cada pieza del modelo y formar grupos con ellas. Esta posibilidad resulta muy útil cuando se está trabajando con muchas partes. Antes de realizar las operaciones de agrupamiento hay que renombrar cada pieza. Para ello debemos irnos a la pestaña "Groups" de las opciones de edición. Observaremos una ventana que muestra una lista de todas las partes de la bionave.



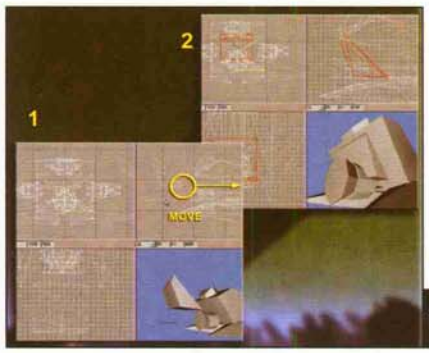
8

A partir de cilindros con tres divisiones, obtenemos los cañones.



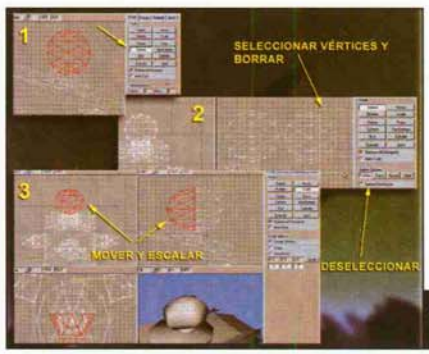
9

Dos piezas importantes, pero muy fáciles de realizar.



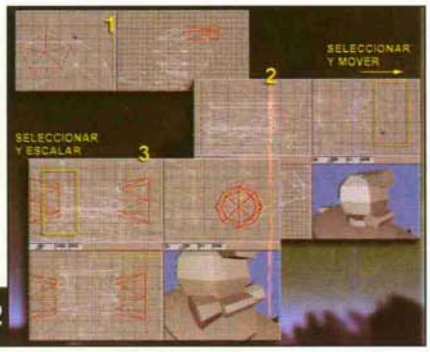
10

Una pieza de detalle para equilibrar la forma del casco de la nave.



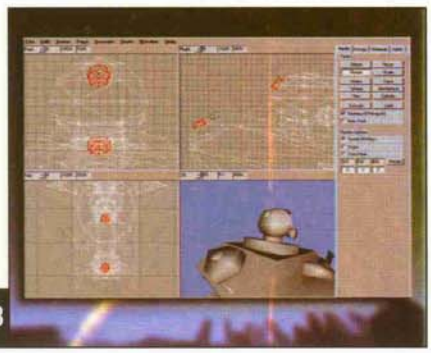
11

Hay que encajar bien el conector al casco mediante un buen escalado.



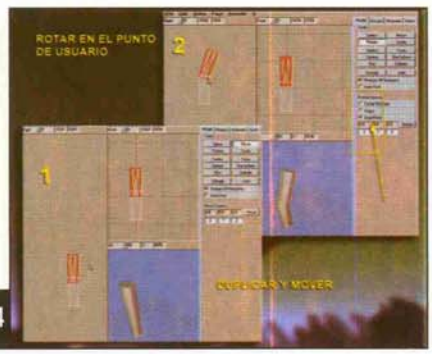
12

La cámara y el depurador de aire son dos piezas importantes para el aspecto humano al casco.



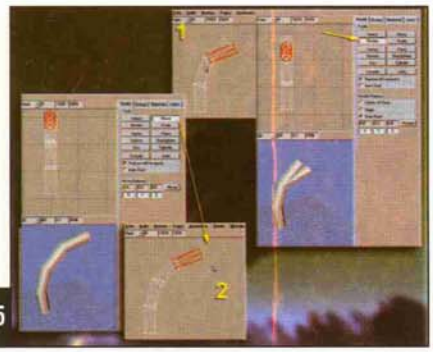
13

Los ajustes del tubo son piezas de detalle.



14

Primera forma de modelar el tubo.



15

La rotación de las piezas activando la opción "User Point" es bastante útil.

De la lista señalamos, por ejemplo, el primer elemento y en "Group" escribimos su nombre en el hueco de texto situado a la derecha del botón "Rename". Una vez realizada esta acción, pulsamos sobre"Rename" para completar el proceso. Cuando tengamos todas las piezas renombradas, el siguiente paso será asignarlas a cada uno de los botones numéricos que aparecen en "Smoothing Groups". Dejamos pulsado el botón "Assign", pero antes de asignar una pieza a un botón, hay que seleccionarlas. La elegimos de la lista y pulsamos sobre el botón "Select" en "Group", luego basta con pulsar el botón numérico que queremos asignar a esa pieza, por ejemplo, el "1". Antes de asignar otra nueva pieza es necesario deseleccionar la anterior pulsando de nuevo en "Select". Para los siguientes elementos realizaremos las mismas operaciones. Si queremos borrar todas las asignaciones pulsamos en "Clear All". En las opciones del apartado "Group", además de seleccionar, podemos ocultar ("Hide") y borrar ("Delete") cualquier pieza del modelo.

GUARDANDO EL MODELO

Antes de salvar el modelo en disco es importante tener claro el uso posterior que se le va a dar. Debemos saber en qué formato trabajaremos y si el modelo estará animado o no. Milkshape3D permite exportar a una multitud más que aceptable de formatos diferentes, englobando todas las necesidades posibles. Desde los ficheros de juegos como: *Quake, Unreal, Half Life* o incluso *Los Sims,* hasta los formatos 3D de las aplicaciones y motores gráficos más populares. En nuestro caso, diseñamos las bio-naves con posibilidad de que dispongan de ciertos movimientos. Aunque Milkshape3D permite la realización de esqueletos por puntos de unión a los modelos y dotarlos de movimiento, utilizaremos la aplicación Character FX para tal fin por disponer de más y mejores opciones. Teniendo en cuenta nuestros propósitos, debemos saber que Character FX importa objetos en formato .3DS, .OBJ (aunque puede importar también modelos en formato ASCII procedente del Milkshape3D -.TXT-). Además, para realizar y exportar el modelo con la animación, es conveniente disponer del objeto en un sólo *mesh* (objeto 3D), sobre todo si queremos exportar a .MD2. Así que nos vemos obligados a reagrupar todas las partes de la bio-nave en una sola antes de salvarlo. Para realizar esta operación, sólo tenemos que seleccionar todas las partes del modelo con "Select All" en el menú "Edit" y pulsar el botón "Regroup" en la pestaña "Groups" del panel de edición. Una vez agrupadas todas las partes, salvaremos el modelo en formato .OBJ. Debemos tener en cuenta que si antes de animar el modelo lo vamos a texturizar, no es conveniente entonces agrupar todas las partes en un solo grupo.

Ya tenemos nuestra bio-nave modelada y lista en el disco duro para completar su desarrollo. Pero antes de animarla vamos a realizar el proceso de texturizado.

NOTA

Si no se va a implementar ningún tipo de animación a la bio-nave, puede ser guardada en formato .3DS.

En el próximo número...

... completaremos el aspecto de la bio-nave de combate pintándola con el programa Deep Paint 3D.

Cómo desarrollar los efectos especiales de sonido (III)

Hemos estudiado cómo grabar, editar y aplicar efectos a nuestros sonidos. En esta entrega, completaremos el aprendizaje de Goldwave realizando operaciones muy interesantes.

EL EVALUADOR DE EXPRESIONES

Goldwave nos permite crear y manipular sonidos por medio de funciones matemáticas, muy útil para desarrollar nuestros propios efectos. Realicemos una práctica para conocer el funcionamiento de esta potente herramienta. Creamos una plantilla nueva de 5 segundos y elegimos la opción "Expresión Evaluator" situada en el menú "Tools". Podemos observar que básicamente se trata de un sistema para escribir fórmulas y darles un nombre. Vamos a fabricar el sonido de un motor. Partiremos de la onda de sonido de la función matemática coseno que varía en altura (eje Y). Matemáticamente la fórmula sería *Y= Cos(PI*F*T)*, donde F es la frecuencia y T el tiempo.

Vamos a implementar esta ecuación pero con una pequeña variación, así que, en la zona "Expresión" escribimos *cos(f*t)*(2*pi)*. En "Variables" cambiamos el valor de la frecuencia "f=" a 80 y el tiempo "t=" a 10. Pulsamos en el botón "Start" y listo. Al reproducir el sonido oiremos un motor. Jugando con la frecuencia obtendremos sonidos más agudos o graves. Por ejemplo, probad a colocar un valor igual a 10000 (Fig. 1).

También podemos crear efectos con esta herramienta, personalizando aún más nuestros sonidos.

TRUCO

En la práctica, este tipo de modificaciones a tan bajo nivel puede ser útil para retocar interferencias, ruidos, picos de nivel o simplemente proporcionar algún tipo de efecto personal al sonido.

EDITAR DIRECTAMENTE LA FORMA DE ONDA

Goldwave permite al usuario actuar directamente sobre la forma de onda. Esta opción posibilita tener una edición más exhaustiva del sonido y realizar cambios al milímetro dibujando con el ratón. Antes de poder actuar sobre la onda del sonido es necesario hacer zoom hasta alcanzar un nivel 2:1 o 5:1 (Fig. 2).

Para realizar una práctica, carguemos un sonido cualquiera que tengamos en el disco o grabemos uno nuevo.

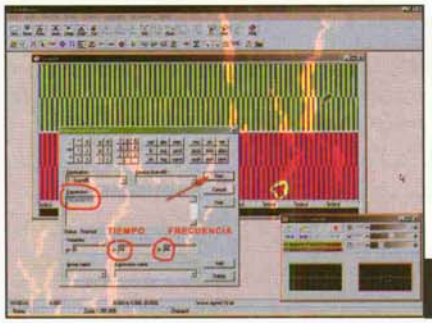
Elijamos una zona del sonido y hagamos zoom hasta alcanzar un acercamiento 5:1. Al pasar el cursor del ratón sobre el dibujo de la onda, éste cambia de forma (una raya horizontal entre dos flechas verticales), indicando que ya es posible modificarla (Fig. 3).

Seguidamente, dibujamos con el ratón mientras hacemos clic con el botón izquierdo.

En la práctica, este tipo de modificaciones a tan bajo nivel puede ser útil para retocar interferencias, ruidos, picos de nivel o simplemente proporcionar algún tipo de efecto personal al sonido. (Fig. 4).

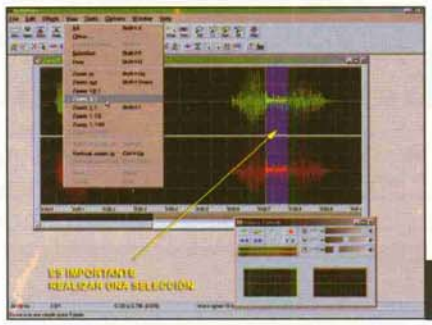
RESAMPLEAR

Este procedimiento nos proporcionará la opción de cambiar la



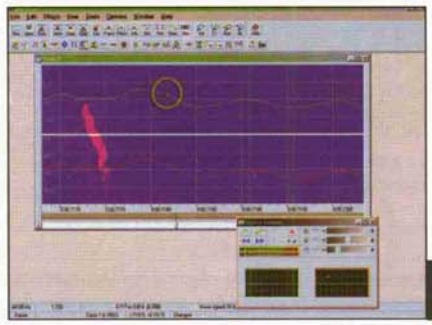
1

Procesos para la creación de un sonido desde cero con el evaluador de expresiones.



2

Para poder modificar manualmente la forma de onda es necesario hacer un zoom hasta un nivel 5:1.

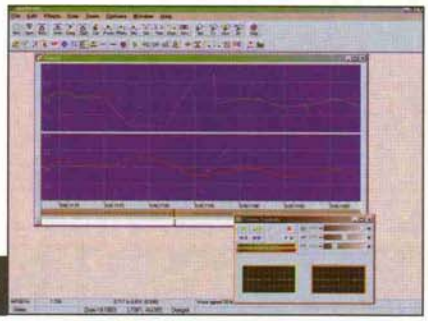


3

Para poder dibujar el sonido, el cursor debe cambiar de aspecto al pasar por encima de la forma de onda.

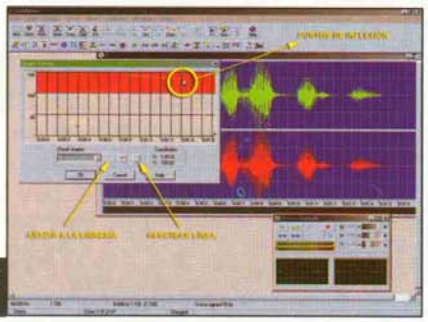
NOTA

Para desplazar la marca de comienzo de selección se pueden utilizar los cursores izquierdo y derecho + SHIFT y para la marca de final, cursores izquierdo y derecho + CTRL + SHIFT.



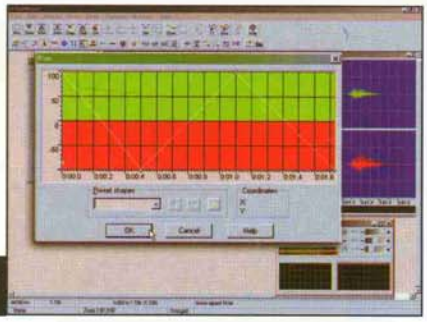
4

Podemos modificar la forma de la onda del sonido por medio del ratón.



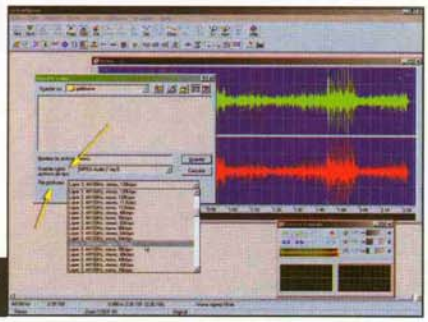
5

Para realizar un volumen personalizado en el sonido, podemos crear y editar cuantos puntos de inflexión deseemos.



6

La gráfica de la panorámica del sonido puede ser modificada con sólo utilizar el ratón.



7

Goldwave puede convertir el formato .WAV a .MP3 y viceversa sin ningún problema.

frecuencia de muestreo de nuestro sonido. Siempre es preferible, en un proceso de grabación y edición, utilizar la frecuencia más alta posible de muestreo. Una vez realizadas todas las operaciones al sonido es cuando se elige su calidad final. Es entonces cuando podemos utilizar esta función. Por ejemplo, si trabajamos con una calidad CD (44 Khz) y queremos obtener un resultado final a 22 Khz elegimos la opción "Resample" en el menú "Effects". Posteriormente, seleccionamos 22050 y "Ok".

Goldwave posee una opción muy interesante que permite reproducir un sonido a distinta velocidad "rate". La podemos encontrar en el menú "Effects" en "Playback Rate". Tiene el mismo funcionamiento que "Resample", pero sólo afecta a la duración y no a la calidad.

AJUSTE PERSONALIZADO DEL VOLUMEN

Hay un procedimiento muy útil para realizar un ajuste más personalizado del volumen del sonido, y es cambiando su envolvente de amplitud a lo largo del tiempo de duración.

En Goldwave encontramos esta función en el menú "Effects" en la opción "Shape" del submenú "Volume". Aparece una gráfica con la medida de volumen en el axis vertical y la duración en el horizontal. Por defecto, aparece el volumen normal en 100, representado por una línea horizontal amarilla. La parte de color roja representa un aumento del volumen y la parte sin color una reducción. Para crear puntos de inflexión en la línea de volumen, es suficiente con hacer clic con el ratón en cualquier lugar de la gráfica. Para mover cualquier punto, nos situamos encima de él y movemos el ratón manteniendo el botón izquierdo pulsado. Si por el contrario deseamos borrar un punto de inflexión, nos situamos sobre él y pulsamos el botón derecho del ratón (Fig. 5).

Como se puede apreciar, esta función es muy útil para dotar a las muestras de cierta expresión y lograr efectos más detallados.

TRUCO

Para cambiar los colores de la ventana de edición tenemos la opción "Colours" en el menú "Tools".

DEFINICIÓN

▸ ENVOLVENTE DE AMPLITUD

La envolvente de amplitud de un sonido es la representación gráfica de la dinámica de su volumen en el tiempo.

AJUSTE PERSONALIZADO DEL PANORÁMICO

De igual forma, podemos actuar directamente en el panorámico del sonido dibujando su evolución en el tiempo a través de una gráfica. Encontramos esta estupenda opción en el menú "Effects" en "Pan", situado en la opción "Stereo". La forma de actuar sobre la gráfica es exactamente igual que en la anterior opción. La única diferencia radica en la gráfica en sí. En ella se representa, con dos colores, cada uno de los canales estéreos del sonido. El color rojo indica el canal derecho y el verde el izquierdo. Podemos entonces dibujar el paso del sonido de un altavoz a otro a nuestro gusto. Probemos, por ejemplo, dibujando la gráfica como se muestra en la figura 6.

Observamos cómo al reproducirlo, va pasando de un altavoz a otro. Jugando con esta opción, podemos obtener sonidos más envolventes y con más cuerpo. El límite está, como siempre, en la imaginación.

En el próximo número...

... realizaremos los sonidos de "Zone of Fighters", empezando por crear los disparos, explosiones y otros efectos.

Gráficos 2D y sprites

Después de imprimir texto y mapa de bits, vamos a continuar aprendiendo más sobre los gráficos en el mundo 2D de Blitz3D.

Estudiaremos otros aspectos del búfer de pantalla, así como el manejo de primitivas gráficas en 2D y cómo trabajar con los píxeles (puntos de color) de la pantalla. Terminaremos explicando el funcionamiento de los sprites dentro de un entorno 3D.

ALGO MÁS SOBRE LOS BÚFERES

En el número 5 explicamos el funcionamiento de los búferes de vídeo y de la técnica del doble búfer. Continuando con este tema, encontramos instrucciones en Blitz3D que nos permiten cargar una imagen directamente a cualquiera de los dos búferes o salvarlos a disco. La función de cargar es *LoadBuffer (búfer, fichero de imagen $)* y la de salvar ya la conocemos: *SaveBuffer (búfer, fichero de imagen $)*.

En Blitz3D también podemos realizar operaciones de gran velocidad con píxeles o puntos de color de la pantalla. Para poder trabajar a este nivel es necesario bloquear el búfer en donde se realizan estas operaciones mediante la instrucción *LockBuffer nombre del búfer*. Para desbloquearlo utilizaremos *UnlockBuffer nombre del búfer*. Una vez bloqueado el búfer, sólo es posible realizar operaciones gráficas con píxeles.

OPERACIONES CON PÍXELS

Para leer un valor de color, es decir, un número que contiene el canal alfa, el color rojo, verde y azul en un punto de la pantalla, se utiliza la función *ReadPixel (X,Y,búfer)*.

Esta instrucción es muy similar a *GetColor X,Y*. La diferencia es que *GetColor* extrae las componentes del color (rojo, verde y azul) y la pasa a las funciones *ColorRed() (componente rojo), ColorGreen() (componente verde)* y *ColorBlue() (componente azul)*.

(Ver funcionamiento en el ejemplo 1 del numero 6: ejemplo6_1.bb).

Existe una función que nos permitirá leer un píxel de la pantalla a mucha mayor velocidad, se trata de *ReadPixelFast (X,Y,búfer)*. Funciona exactamente igual que su hermana *ReadPixel()*, sin embargo es conveniente no olvidarse de bloquear el búfer antes de utilizarla y desbloquearlo después de su uso. Si no lo hacemos así, la función no funcionará y además podría provocar el cuelgue del sistema.

De igual forma podemos escribir un píxel en pantalla con las funciones *WritePixel (X,Y, búfer)* y *WriteFastPixel(X,Y,búfer)* (ver "ejemplo7_1.bb").

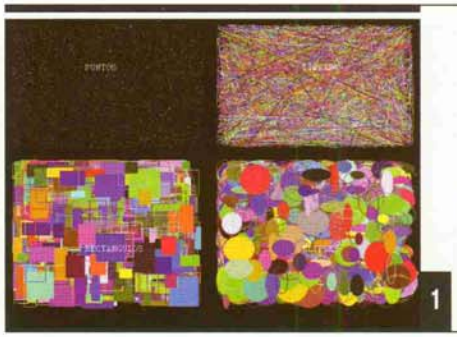
También podemos copiar un píxel de un lugar de la pantalla a otro con las instrucciones:

```
CopyPixel (X origen, Y origen,
búfer origen, X destino,
Y destino, búfer destino)
```

y su equivalente, con más rapidez de resultados:

```
CopyFastPixel (X origen, Y origen,
búfer origen, X destino,
Y destino, búfer destino)
```

Antes de terminar, comentar que para cambiar el color de todas las operaciones de dibujo se utiliza la instrucción *Color Rojo, Verde, Azul*.



Primitivas 2D que puede generar Blitz3D.

PRIMITIVAS 2D

Es interesante también el uso de primitivas gráficas 2D en los videojuegos. Nos referimos a los puntos en la pantalla, líneas, cuadrados o círculos (ver "ejemplo7_2.bb") (Fig. 1).

Para dibujar un punto en la pantalla utilizaremos la instrucción *Plot X,Y*. Para dibujar una línea:

```
Line X comienzo de línea,
Y comienzo de línea, X final
de línea, Y final de línea
```

NOTA

Podemos también bloquear el búfer antes de usar *ReadPixel()* y *WritePixel()* para realizar las operaciones instantáneamente. Sin olvidar que siempre hay que volver a desbloquearlo una vez acabadas estas operaciones.

NOTA

Para saber la memoria de video total de la tarjeta gráfica usaremos la función *TotalVidMem()* y para saber la que queda *AvailVidMem()*.

Si lo que deseamos es dibujar un rectángulo usaremos:

```
Rect X,Y, Longitud, Altura,
1 (relleno) / 0 (transparente)
```

Y para dibujar un círculo o elipse utilizaremos:

```
Oval X,Y, Longitud, Altura,
1 (relleno) / 0 (transparente)
```

OTRAS FUNCIONES GRÁFICAS

Vamos a estudiar algunas funciones muy interesantes que Blitz3D posee para el tratamiento gráfico de la imagen, que nos ayudarán a crear efectos de scroll o sistemas de visualización 2D muy utilizado en la mayoría de los videojuegos.

CREANDO VENTANAS DE VISUALIZACIÓN

Con esta función podemos definir el área de la pantalla que se visualizará; es decir, crea una ventana sobre el fondo y sólo se verá el fondo a través de ella, el resto es ignorado.

Un esquema de ejemplo de interfaz de juego utilizando ventanas.

Utilización de ventanas móviles para mostrar porciones del fondo.

Este tipo de procedimiento visual es muy utilizado en juegos donde la acción se desarrolla en una porción de la pantalla, dejando libre el resto para otros tipos de gráficos, como puede ocurrir en algunos juegos de Rol (Ver Fig. 2).

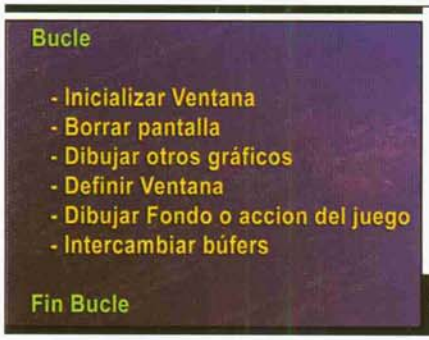
Aunque el concepto parece simple, la utilización de esta función es algo confusa. Así que vamos a ver detenidamente cómo funciona y cómo debemos utilizarla.

```
ViewPort Coordenada X,
Coordenada Y, Tamaño horizontal,
Tamaño vertical
```

Las coordenadas X e Y indican dónde está situada la esquina superior izquierda de la ventana y el tamaño horizontal y vertical hasta donde se extenderá. Es importantísimo tener en cuenta que es necesario borrar una ventana creada en el caso de definir otra nueva. Para realizar esta operación basta con definirla como si fuera la pantalla completa; es decir, si definimos un modo gráfico de 800 x 600, debemos escribir "ViewPort 0,0,800,600" para borrar la vista.

En el ejemplo "ejemplo7_3.bb" (Fig. 3) se puede observar un efecto realizado con esta función, en donde se crea e inicializa continuamente una ventana de visualización, generando así una ventana movible sobre un fondo fijo.

La siguiente cuestión que nos podemos plantear es cómo conseguimos tener independientemente una vista definida en donde se muestra un fondo (zona de juego) y unos gráficos que representen, por ejemplo, la puntuación u otra actividad (un buen ejemplo de esta situación la podemos encontrar en cualquier juego de acción donde se muestre en una esquina de la pantalla un radar). Pues bien,



Esquema del orden a seguir para visualizar fondos y otros gráficos utilizando ventanas.

para resolver este problema debemos establecer un orden específico a la hora de visualizar los gráficos. En la figura 4 se muestra el esquema de este orden.

En el ejemplo "ejemplo7_4.bb" se puede observar cómo funciona este procedimiento.

CREANDO UN SCROLL PARALLAX

La implementación de sistemas de scroll de pantalla en un juego es una tarea que puede tomar muy diversos caminos. La manera más sencilla, aunque no la más rápida y eficaz, es sin duda el movimiento completo de los fondos. Generalmente esta técnica es utilizada para scrolls cíclicos, es decir, el fondo se repite continuamente, por ejemplo, cuando sale por la izquierda va apareciendo por la derecha de la pantalla. Blitz3D posee algunas funciones específicas para este tipo de procedimientos: "TileBlock" y "TileImage":

```
TileBlock Imagen [,Coordenada X,
Coordenada Y, fotograma de la
imagen a usar]
TileImage Imagen [,Coordenada X,
Coordenada Y, fotograma de la
imagen a usar]
```

Estas funciones realizan un tileado (repetición) de la imagen utilizada para los campos de scroll, además de poderlas mover. La única diferencia entre una función y otra es que "TileBlock" ignora

el color utilizado para la transparencia y "TileImage" no. En el ejemplo "ejemplo7_5.bb" se muestra un ejemplo muy sencillo de scroll múltiple (parallax) utilizando estas dos funciones.

Es importante tener en cuenta el orden de utilización de "TileBlock" y "TileImage" para lograr el efecto deseado.

SPRITES

Sabemos que Blitz3D trata a los elementos que forman el mundo 3D (luces, terrenos, modelos) como entidades. Pues bien, los sprites en Blitz3D son también entidades planas de forma cuadrada o rectangular con sólo dos polígonos. Por defecto, este objeto plano siempre mira hacia la cámara, dando la sensación de que es un objeto en 3D y además es tratado como si lo fuera.

CREANDO SPRITES

El uso de sprites es muy útil para simular explosiones, humo o disparos en un entorno 3D. Al ser tratado como un objeto 3D o entidad es posicionado en la pantalla a través de las coordenadas X, Y y Z. En el ejemplo 3 ("ejemplo7_6.bb") del CD creamos un sprite y lo colocamos delante de la cámara. Además, con la tecla "1" activamos o desactivamos el modo *wireframe* (alámbrico, sin textura). De esta forma podremos observar la forma y los polígonos que forman un sprite. Para crear un sprite nuevo, sólo es necesario una variable que lo contendrá (Fig. 5):

```
Sprite = CreateSprite ()
```

Como comentamos, esta entidad llamada Sprite es tratada como un objeto 3D, así que podemos aplicarle los mismos efectos y posicionarla en el entorno 3D. Luego, si queremos, le podemos aplicar una textura para darle imagen. Pero hay una forma de realizar estas dos operaciones a la misma vez, y es con la instrucción:

```
LoadSprite (imagen_textura$
[, modo de aplicar textura]
[, entidad madre])
```

Este comando te permite cargar una imagen para crear un sprite con ella. En realidad, lo que hace es crear un sprite y asignarle como textura la imagen cargada. Además, opcionalmente es posible asignar la textura de diferentes modos (opción "[,modo de aplicar textura]"). Así disponemos de las siguientes opciones (ver "ejemplo7_7.bb") (Fig. 6):

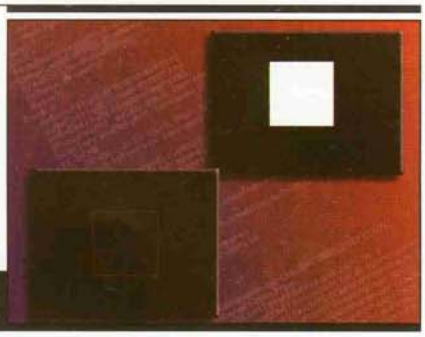
- **Valor 1: Color del mapa.** Es el valor por defecto y se muestra como es.
- **Valor 2: Canal Alfa.** Si la textura tiene canal alfa, las zonas alfas de ésta se harán transparentes.
- **Valor 4: Máscara de color.** Todas las áreas de la textura con color negro (0,0,0) no se dibujarán.
- **Valor 8: Mipmapped.** Se muestra una versión de la textura con baja calidad a grandes distancias.
- **Valor 16: Ajuste U** (horizontal) de la textura para evitar enrollamientos.
- **Valor 32: Ajuste V** (vertical) de la textura para evitar enrollamientos.
- **Valor 64:** Se utiliza para realizar un mapeado del entorno con brillo.

Generalmente, las más comunes suelen ser los valores 1 y 2, sobre todo si utilizamos sprites para representar fuego, humo, etc.

La siguiente opción ([,entidad madre]), permite asignar al sprite una entidad madre, esta cualidad es muy útil, ya que si movemos o escalamos la entidad madre el sprite lo hará también. Sin embargo, si movemos el hijo (sprite), no afectará a la entidad madre. Por ejemplo, el protagonista del juego (entidad madre) lleva una piedra (entidad hijo, sprite) en la mano. Cuando el protagonista se mueve la piedra se moverá con él y seguirá en la mano. Al arrojar la piedra, se desplaza el sprite "piedra", pero el protagonista no se mueve con ella, se queda en la misma posición.

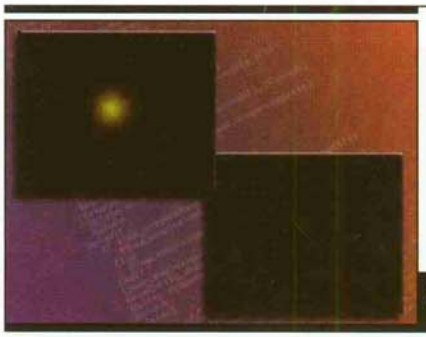
RECORDATORIO

Un sprite es una imagen (generalmente de pequeño tamaño) usada para gráficos animados o simplemente para un icono.



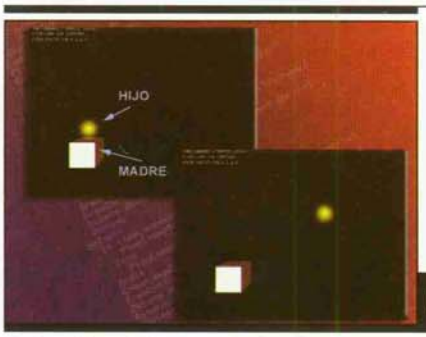
5

En la imagen se puede comprobar que un sprite está formado por dos polígonos planos.



6

Un ejemplo de textura para sprite con la opción de máscara activada.



7

En la salida del ejemplo 5.bb, se puede comprobar la dependencia de una entidad hijo hacia su entidad madre.

En el ejemplo 5 ("ejemplo7_8.bb") del CD se puede observar cómo al mover el cubo (entidad madre) arrastra al sprite (entidad hijo) consigo, mientras que si se mueve el sprite, el cubo se mantiene inmóvil (Fig. 7):

```
...
cubo=CreateCube()
sprite=LoadSprite("c:\particle
.bmp",1,cubo) ←Sprite, modo
texturizado, padre/madre
PositionEntity Sprite,0,2,0←
PositionEntity cubo,0,0,15←
...
```

Al crear el sprite, le asignamos como madre el cubo. Posicionamos el sprite arriba del cubo para que se pueda ver. Observar que sólo tenemos que desplazar la coordenada Y del sprite y no la Z, ya que al ser hijo del cubo, posicionando éste se posicionará el sprite con él.

MANIPULANDO SPRITES

Aunque pueden ser controlados como una entidad, los sprites tienen algunos comandos propios para rotarlos o cambiar su tamaño (ver "ejemplo7_9.bb") (Fig. 8):

Con *RotateSprite sprite, angulo de rotación #* podemos rotar un sprite desde su eje y con *ScaleSprite sprite, escala horizontal#, escala vertical#* podemos cambiar su tamaño.

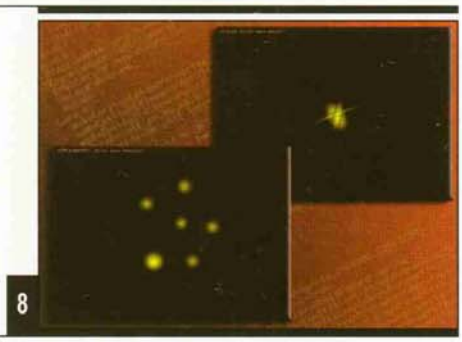
Para cambiar el eje de rotación del sprite utilizaremos la instrucción:

```
HandleSprite sprite,
desplazamiento horizontal#,
desplazamiento vertical#
```

Esta instrucción también nos sirve para desplazar el sprite (ver "ejemplo7_10.bb") (Fig. 9).

RECORDATORIO

Blitz3D define a todos los elementos 3D como entidades. Este sistema facilita un método de asignaciones muy útil que permite controlar el estado, el movimiento o las colisiones de luces, cámaras o superficies como si fueran objetos.



8

Algunos efectos interesantes con las opciones de escalado o desplazamiento de un solo sprite.

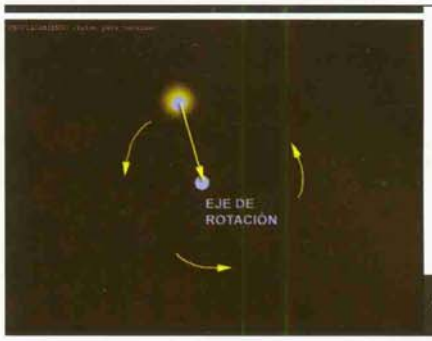
MODOS DE VISTA

El problema que surge cuando se visualiza un objeto plano, como un sprite, en un entorno 3D, es cómo controlar su orientación con respecto a la cámara. Por ejemplo, supongamos que tenemos un juego; en él se muestra un terreno en 3D con muchos árboles. Estos árboles, para que el juego tenga un buen rendimiento, son realmente sprites, es decir, un objeto plano con el dibujo de un árbol (textura). Si orientamos los árboles siempre hacia la cámara, podremos ocultar que en realidad se trata de un dibujo plano y no de un objeto en 3D. Esta técnica fue la utilizada en los antiguos juegos en 3D, en donde los enemigos y objetos de decoración eran sprites orientados hacia la cámara.

Blitz3D permite controlar de cuatro formas diferentes de orientación con respecto a la cámara. Para activar esta operación se utiliza la instrucción:

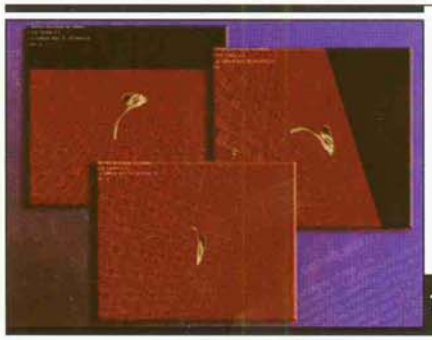
```
SpriteViewMode sprite,
modo de orientación.
```

Por defecto se encuentra activado el modo 1, en el cual el sprite siempre está mirando a la cámara. El modo 2 se mantiene al sprite libre de orientación. Con el modo 3 se consigue una orientación hacia la cámara, pero no cuando ésta se inclina, y en el modo 4 se consigue una orientación completa con todos los movimientos y giros de la cámara.



9

La función HandleSprite puede desviar el eje de rotación de un sprite.



10

Distintos comportamientos en el modo de vista de un sprite frente a la cámara.

En el ejemplo 8 ("ejemplo7_11.bb"), se puede observar el funcionamiento de esta importantísima función (Fig. 10).

A los sprites se les puede aplicar todos los controles, como a cualquier otra entidad. Así, es posible detectar colisiones con ellos o entre ellos, moverlos por el entorno 3D, manipular sus características, aplicar efectos y obtener información de su estado.

En el próximo número...

... explicaremos todas estas operaciones, comunes entre todas las entidades. Aprenderemos todo lo necesario sobre este sistema utilizado por Blitz3D y así prepararnos más a fondo para manipular el mundo 3D y todas sus funciones.

Editores de audio. Sound forge 6.0 (I)

En este número empezamos una serie de tutoriales básicos dedicados a los editores de audio más populares del mercado.

Para comenzar, hemos querido tener aquí al que quizás sea uno de los más potentes, Sound Forge 6.0. Por lo general, la forma de usar los editores de audio es muy similar, así que nos centraremos sólo en las nuevas posibilidades que nos podamos encontrar en cada uno de ellos.

SOUND FORGE 6.0

Sound Forge es de la desarrolladora Sonic Foundry, especializada en aplicaciones para el control de audio. Este programa siempre se ha caracterizado por tener una interfaz de usuario muy cómoda, flexible e intuitiva desde la primera versión. Es ampliamente utilizado en el mundo amateur y profesional por su versatibilidad y potencia. Una característica importante es la posibilidad de poder tener varios sonidos cargados a la vez en ventanas diferentes y trabajar con calidades de hasta 32 bits a 192 Khz. Pero quizás lo que ha distinguido este programa de los demás es su velocidad de trabajo en procesos de edición como cortar, pegar o mover y en la posibilidad de obtener previos en tiempo real de los procesos con efectos antes de aplicarlos. Además, permite visualizar ficheros de vídeo para poder así sincronizar, por ejemplo, el audio para una película. Puede trabajar con multitud de formatos de audio como .WAV, .MP3, .OGG (OggVorbis), .AIF, .SND, etc.

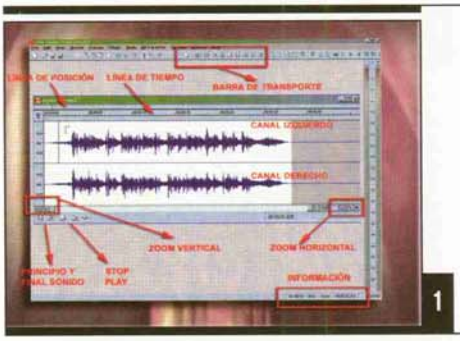
TRUCO

Para mover la línea de tiempo podemos utilizar los cursores derecha e izquierda o CTRL + dichos cursores. Además , para realizar una selección podemos mover la línea de tiempo con el método anterior pero manteniendo pulsada la tecla SHIFT.

CARGAR, REPRODUCIR Y EDITAR UN SONIDO

Podemos cargar un sonido de dos formas diferentes. Una manera sería sólo para reproducirlo, sin posibilidad alguna de editarlo. Para ello sólo tenemos que activar la casilla "Open as read-only" en la ventana "Abrir". Si activamos esta casilla, cargaremos el sonido y podremos entonces editarlo. Una vez cargado, se abrirá una ventana con la gráfica del sonido. En la figura 1 se describe cada uno de los elementos que componen la ventana de edición.

Para reproducir el sonido, podemos pulsar en el icono , la barra espaciadora (también para parar) o bien desplazar el puntero del ratón haciendo clic sobre la barra situada sobre la línea de tiempo (Ver Fig. 1). Para seleccionar una porción del sonido es suficiente con utilizar el ratón, manteniendo pulsado el botón izquierdo mientras desplazamos el puntero. El principio y final de selección se pueden mover, independientemente, también con el ratón. Para seleccionar sólo uno de los canales es suficiente con hacer clic en la zona más externa del canal.



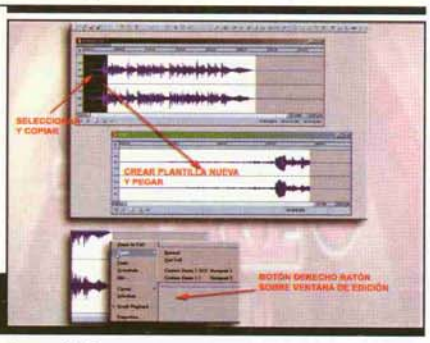
Descripción de los elementos del la ventana de edición.

Cuando pulsamos el botón derecho del ratón encima de la ventana de edición aparecerá un menú flotante con diferentes opciones dependiendo de si hemos seleccionado o no una parte del sonido. En caso de no tener ninguna selección, podemos hacerlo en la opción "Selection" de dicho menú.

Para cortar un trozo del sonido basta con seleccionarlo y pulsar CTRL + X. Para insertar o copiar una porción de sonido de un lugar a otro seleccionamos lo que queremos copiar, pulsamos CTRL + C y llevamos la barra de posición, haciendo clic con el ratón, al lugar donde queramos colocarla. Posteriormente pulsamos CTRL + V. Es posible hacer esta misma opera-

NOTA

Si pulsamos el botón derecho del ratón fuera de la ventana de edición aparecerá un menú flotante con acceso rápido a las funciones de archivo y configuración del programa.



2

Procedimiento para crear un nuevo sonido a partir de una selección.

ción en otro sonido diferente u otra ventana de audio nueva. Seleccionamos un trozo de sonido y copiamos con CTRL + C. Pulsamos en "File \ New". Se abrirá una nueva ventana en blanco. Hacemos clic en cualquier lugar de esta nueva ventana y pulsamos CTRL + V. La porción seleccionada del primer sonido la hemos copiado a la nueva plantilla (Fig. 2).

Para realizar las operaciones de zoom disponemos de multitud de opciones. Bien mediante los iconos lupas situados a ambos extremos de la ventana de edición, a través del menú flotante de edición o por medio de las opciones de zoom del menú "View" (Fig. 2).

TRUCO

Podemos alejarnos o acercarnos a la gráfica del sonido de una forma más rápida por medio de los cursores. Cursor arriba para acercarnos y cursor abajo para alejarnos.

3

Diferentes maneras de cambiar el formato de un sonido.

4

Descripción de los elementos de la ventana de grabación.

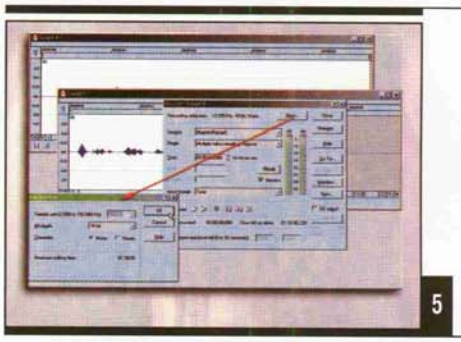
GRABAR UN SONIDO

Vamos a explicar de una forma breve el procedimiento para grabar un sonido desde un micrófono externo. Supongamos que queremos grabar una voz a 22 Khz, mono y calidad de 16 bits. Creamos una nueva plantilla en "File \ New". A continuación, debemos cambiar el formato de dicha plantilla. Hay varias formas, una primera, a través del menú flotante de edición en la opción "Properties" y otra, pulsando el botón derecho del ratón sobre la información correspondiente del fichero y eligiendo la opción deseada (Ver Fig. 3).

Una vez elegido el formato, pulsamos en el icono y aparecerá una ventana de opciones cuya descripción se muestra en la figura 4.

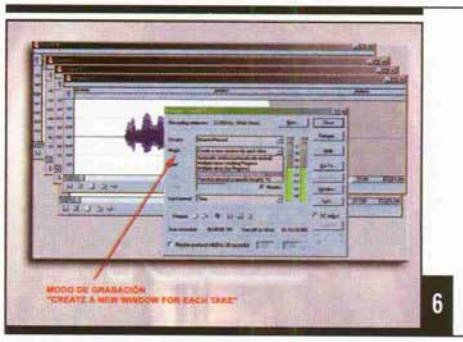
Dentro de ésta, el botón "New", nos permite abrir una nueva plantilla para grabar otro sonido sin tener que salir de la ventana "Record" (Ver Fig. 5).

Además nos permitirá crear el nuevo sonido en otro formato. Sin embargo, la mejor opción, si queremos hacer grabaciones continuas con el mismo formato, es elegir "Create a new window for each take" en el modo ("Mode") de grabación. Cada vez que pulsemos el botón de grabar se creará una nueva plantilla automáticamente para alojar la nueva grabación. Este modo de trabajar ahorra mucho tiempo y esfuerzo y se consiguen grabaciones más uniformes (Ver Fig. 6).



5

Procedimiento para grabar nuevos sonidos con diferentes formatos desde la ventana de grabación.



6

Procedimiento para la creación y grabación continua de sonidos con el mismo formato.

TRUCO

Para grabar un sonido nuevo, hemos de asegurarnos de hacerlo en una habitación silenciosa, para que no se oiga ningún ruido de fondo. Es conveniente también no acercarse demasiado al micrófono.

En el próximo número...

... aprenderemos a crear sonidos nuevos, realizar "loops", listas de selecciones, y haremos una práctica creando el sonido de una explosión desde cero.

Evolución técnica de los arcades

La estructura de los arcades se basaba en más o menos grandes mapeados que conformaban todos los niveles del juego.

La evolución técnica de estos juegos pasa por la manera de representar este mapeado en pantalla, además, claro está, de la mejora de los gráficos, animaciones y sonidos. Por último, cabe destacar la revolución técnica que supuso el paso de la segunda a la tercera dimensión para este tipo de juegos.

Nos encontramos, de este modo, diferentes estilos de juegos de plataformas según la forma de representar el mapeado: pantalla fija, scroll de pantalla simple y scroll de pantalla múltiple.

JUEGOS DE PLATAFORMAS DE PANTALLA ESTÁTICA

En los comienzos del desarrollo de juegos de plataformas para PC, era complicado el uso del scroll de pantalla por limitaciones técnicas. En su lugar, se optó por el dibujo de las pantallas una a una, las cuales iban apareciendo a medida que el protagonista llegaba a los bordes de la misma o bien cruzando una puerta o un teletransporte. Estos juegos para PC adoptaron la forma de la mayoría de las producciones para los microordenadores más populares como el ZX Spectrum, MSX, Atari o Amiga. A veces, los mapeados de estos juegos eran grandes dibujos divididos en cuadrículas, las cuales hacían referencia a una pantalla.

Era muy común el uso de *tiles* para representar cada pantalla, ya que no se disponía de mucha cantidad de memoria para almacenar muchos gráficos diferentes. Las pantallas se formaban a partir de pequeños gráficos colocados estratégicamente, como si de un mosaico se tratara, para formar las paredes, suelos, etc.

Prácticamente todos los títulos desde 1986 hasta 1990 utilizaban esta técnica. Destacamos las producciones españolas *Livingstone Supongo* (Topo Soft, 1989) o *Goody* (Opera Soft, 1987), así como la serie de *Rick Dangerous* (Core, 1989-90).

EL SCROLL DE PANTALLA

El uso del scroll de pantalla se hizo cada vez más popular, al dar un aspecto visual más atractivo al juego y generar un tipo de jugabilidad más dinámica. Con este sistema, los mapas iban cruzando la pantalla en la dirección en que se movía el protagonista. Era un recurso muy común entonces el uso de los mapeados por *tiles*. Esta técnica ofrecía la posibilidad de generar grandes escenarios con pocos gráficos, ya que los *tiles* estaban diseñados tal que unidos formaban un patrón ampliable sin fin.

Fue el juego *Super Mario Bros* el que prácticamente introdujo esta técnica para la consola de Nintendo. Por entonces, el PC no podía visualizar más de 16 colores a la vez y no era demasiado potente para representar decentemente un scroll por *tiles* en un monitor. Sin

1

Dos ejemplos del mapeado de un juego de plataformas.

2

El mapeado de "Cauldron 2" disponía de cientos de pantallas que unidas formaban un castillo.

3

Goody y la serie Rick Dangerous eran dos claros ejemplos de plataformas de pantalla estática.

4 *El Super Mario Bros de la consola Nintendo introdujo el concepto de plataformas por tiles que luego fue adoptado por multitud de juegos de PC.*

5 *Risky Woods fue un magnífico ejemplo de plataformas con doble scroll.*

6 *El Scroll múltiple trajo profundidad y dinamismo a los escenarios de los juegos de plataformas.*

7 *Pandemonium constituyó el primer paso de los juegos de plataformas en 2D al mundo 3D.*

embargo, sale al mercado una serie de juegos llamados *Commander Keen* (1990) de una empresa nueva llamada ID Software, que utilizaba scroll por tiles. Muchos fueron lo que le siguieron, sobre todo arropados por nuevas tecnologías que poco a poco alimentaban al popular compatible del IBM PC.

SCROLL PARALLAX

Ya en 1991 se empezó a trabajar masivamente con técnicas de scroll más complejas. Por ejemplo, se utilizó un doble scroll para simular el movimiento del cielo como en *Risky Woods* (Dinamic, 1992) o *Prehistorik 2* (Titus, 1992).

Y poco a poco se pudieron mezclar varios scrolls a la vez. Nace así el scroll *parallax* o de paralelaje, consistente en mezclar varios planos de decorado desplazándose a distinta velocidad para simular profundidad.

Se mezclaban fondos móviles que simulaban el cielo o montes y mapeados por *tiles* en los primeros planos de scroll. Se lograba así un realismo extremo similar a las recreativas, y pronto todos los títulos del género optaron por usarlo.

Aunque estas formas de representación iban unidas al avance técnico, significaban también un estilo de juego, y que títulos que no entendían de sistemas o modas, mantuvieran uno u otro método a favor de la jugabilidad. Es el caso de juegos como *Another World* o *Flashback*, que si hubieran optado por el paso de pantalla mediante scroll habrían perdido todo su encanto estratégico.

EL PASO AL 3D

La evolución tecnológica, una vez más, dio la oportunidad a los desarrolladores de arcades de plataformas de investigar nuevas tendencias de visualización. Fue el caso del estancamiento que sufrió la eterna e inamovible segunda dimensión frente al mundo 3D. Este fenómeno ocurrió, sin embargo, en las consolas con el juego *Crash the Bandicoot* y *Mario 64* de manos de Nintendo. Para el PC se empezó a realizar títulos utilizando el 3D con resultados bastantes aceptables para la época. Un buen ejemplo y quizás el primero de este tipo lo tenemos en la conversión del juego *Pandemonium* (*Crystal Dynamics*, 1996) de PlayStation para el PC. Soportaba resoluciones de hasta 640 x 480 y gozaba de todo lujo de detalles gráficos. Realmente se trataba de un juego en 2.5 D (dos dimensiones y media) ya que el personaje se movía como si fuera en 2D, es decir, como en todos los juegos de plataformas (correr hacia delante y atrás y saltar), pero todo el entorno era en auténtico 3D donde la cámara seguía la acción automáticamente.

En otros títulos de calidad, como el clásico *Rayman*, se optó por seguir la serie adaptando el juego a los nuevos tiempos 3D. Surgió entonces el maravilloso titulo *Rayman 2: The Great Scape* (*Ubi Soft*, diciembre del 99).

Hoy día, se realizan pocos juegos de plataformas como antaño. Quizás *Abe´s Oddysee* y su continuación *OddWorld Abe´s Exoddus* (*Oddworld Inhabitants*, 1998) (posiblemente los mejores plataformas de la historia del PC), sean un claro ejemplo de que este género tiene mucha guerra que dar todavía.

En el próximo número...

... hablaremos de un tipo de arcade que está siempre presente en la vida de cualquier jugador: los matamarcianos o *shoot´em up*.

Cuestionario Videojuegos 7

Preguntas

1. ¿Cómo podemos cambiar el color de un píxel de la pantalla?
2. Describe la función que nos permite orientar un sprite hacia la cámara de diferentes maneras.
3. Escribe un grupo de sentencias que nos permita mover la cámara mediante el ratón.
4. ¿Qué funciones nos permitirán, por ejemplo, mover un objeto por un terreno utilizando el ratón?
5. ¿Cómo podemos unir dos piezas en MilkShape 3D?
6. ¿En qué formato podemos guardar el modelo para utilizarlo en CharacterFX?
7. ¿Para que nos puede servir utilizar la función de "resamplear" en GoldWave?
8. ¿Cómo podemos conseguir que el sonido pase de un altavoz a otro?
9. En el programa Sound Forge 6.0, ¿qué modo de grabación debemos usar para realizar grabaciones diferentes con el mismo formato sin salir de la ventana de grabación?
10. En Sound Forge 6.0, ¿cómo podemos crear un sonido desde cero sin realizar ningún tipo de grabación?

Respuestas al cuestionario 6

▷ **1.** La manera más rápida sería aplicando el comando "Replace":

```
frase$="Mi coche es verde"
Replace(frase$,"es ","")
```

Sustituimos "es " por nada.

▷ **2.**

```
ImageCollide (imagen_raton, posX_raton, posY_raton, 0,
imagen_boton, posX_boton, posY_boton, 0)
```

▷ **3.** Por orden irían: logotipo distribuidor, logotipo desarrollador, logotipo del juego, argumento del juego. Necesarias serían el logotipo del distribuidor y del desarrollador.

▷ **4.** Un menú se puede dividir en: opciones gráficas, opciones de audio, opciones de control y opciones de juego.

▷ **5.** Moviendo las líneas discontinuas que marcan el comienzo y final de la letra.

▷ **6.** Para las edificaciones se utilizará el formato .3DS y para la bionave de combate se utilizará .3DS y si le aplicamos animación, . MD2 o . B3D.

▷ **7.** Aplicando un proceso de normalizado, es decir, aumentando al máximo el volumen sin llegar a saturar.

▷ **8.** Por medio del evaluador de expresiones.

▷ **9.** Una forma es en el "Track info" de la ventana de arreglo y otra a través del editor midi GM/GS/XG.

▷ **10.** La técnica de la cuantización.

Contenido CD-ROM 7

▶ AUDIO

■ Dance eJay 4.0
Realiza tu propia música de baile y sus correspondientes efectos visuales.

■ Beat 2000 Online 1.07
Podrás crear tu propia música para PC, aunque no tengas ni idea de composición.

■ Midi Quartet 1.0
Compón, graba, edita y reproduce gracias a este compositor de 12 canales.

■ Ear Trainer 1.0
Entrena tu oído musical con esta divertidísima pero útil herramienta.

■ Jambient 0.9
Usando el ratón o un joystick podrás componer asombrosas melodías.

■ Soundtrack Producer 1.01
Ponle banda sonora a tus vídeos gracias a este programa.

■ Soundforge 6.0

La aplicación que tratamos en el tutorial de este número. Potente y sencilla en su uso.

▶ DISEÑO 2D

■ NewView Graphics' File Viewer 6.0
Podrás ver imágenes en muy distintos formatos gracias a este programa.

■ Pixia 2.5
Edita imágenes fácil y cómodamente usando esta aplicación.

■ Adobe Type Manager 4.1
Demo del popular programa de Adobe de manejo de fuentes, en su versión en castellano.

■ TextPipe Pro 6.3
Añade complejos y vistosos efectos de texto sin necesidad de programar.

■ Ice Color 1.1
Potente herramienta para corregir colores y optimizar tus imágenes.

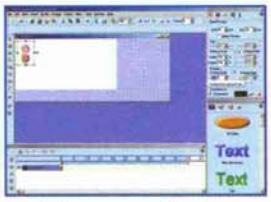
■ PhotoPlus 5
Crea y edita tus propios gráficos con la ayuda de esta completo programa.

▶ DISEÑO 3D

■ Xibit3D 1.0
Con esta aplicación podrás crear interesantes efectos 3D.

■ 3D Movie Maker
Crea fácilmente películas en 3D con este programa y da rienda suelta a tu creatividad.

■ 3D Gif Designer 2.2



Si en algún momento quieres hacer una web para mostrar tu juego, esta herramienta te será de gran ayuda.

■ 3D Textures and Wallpapers
Una colección muy útil de texturas en tres dimensiones.

■ StyleSkin 2.0
Diseña y renderiza interfaces gráficas en dos o tres dimensiones.

▶ PROGRAMACIÓN

■ Mioplanet Pixel Ruler 133
Gracias a esta utilidad podrás saber la posición exacta de un objeto dentro de la pantalla.

■ Power Render SDK 4.0

Esta potente solución para desarrolladores de juegos tiene un montón de opciones y herramientas.

■ LocWise 2.0
Kit de desarrollo que te permite proveer soporte multilingüe gracias a packs de lenguaje.

■ UCanCode Form ++ Class Library 4.0
Completa librería, indicada sobre todo para el dasarrollo de aplicaciones con gráficos. Incluye Visual Studio 6.0

■ REBOL 2.5.0.3.1.
Aplicación para programar multi-plataforma.

■ EmEditor 3.28
Este editor soporta la sintaxis de un gran número de lenguajes, y además es compatible con Windows XP.

▶ JUEGOS

■ Abandon Loader 0.8b
Excelente utilidad para configurar y reproducir juegos *abandonware* de DOS.

■ Lemmings 3D

Versión 3D del popular juego en el que debes llevar a los pequeños lemmings a un lugar seguro.

■ SimCity 2000 Network Edition 1.0
Maravilloso juego apto para todas las edades.

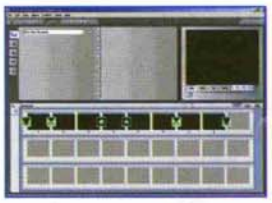
■ Walk Around 3d 1.0
Ejemplo hecho con Blitz3D.

■ Unreal Tournament 2003
Demo del popular juego 3D en su nueva versión que sin duda te entusiasmará.

■ Zone of fighters
De nuevo incluimos la nueva versión del juego, para que no te pierdas detalle.

▶ VÍDEO

■ Pinnacle Studio 7



Potentísimo programa muy sencillo de usar para editar vídeo y aplicarle impresionantes efectos avanzados.

■ Camtasia 3.0.2.
Graba, edita y reproduce vídeo de calidad de un modo muy profesional.

■ GlobFX Player 1.0.9
Herramienta muy completa con la que podrás ver y crear tus propios clips.

■ Windows Media Services 4.1.00
Descárgate los vídeos de mejor calidad gracias a esta interesante utilidad.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable. Además, podrás encontrar dos *plugins* para Milkshape 3D y Character FX, para poder utilizar el formato .B3D.

Codificación del juego, descripción y esquema del programa

Una vez completado el diseño de todos los elementos que componen el juego, el siguiente paso es implementarlos mediante programación para obtener un resultado tangible. Este proceso, por lo general, es muy laborioso y requiere de muchas pruebas. Un buen método para evitar el continuo ensayo-error es planificar todo el código mediante pseudocódigo. Pero antes, vamos a recordar algunos conceptos y premisas, necesarias poder acometer con más control nuestra labor.

(■) CONCEPTOS

Aunque la programación de ordenadores requiere el cumplimiento de ciertas reglas, generalmente impuestas por el lenguaje utilizado, constituye una disciplina con muchos matices artísticos y lógicos. Esta disciplina está repleta de algoritmos para resolver los diferentes problemas que plantea la obtención de un resultado preciso. Estos procesos están formados por un determinado conjunto de instrucciones y estructuras que, dependiendo de cómo se apliquen, se llegará al resultado final de una forma u otra. En las combinaciones que se puedan usar estará la clave para escribir un buen código que aproveche todos los gráficos y sonidos de forma aceptable para el desarrollo del juego. Hay que buscar siempre la máxima optimización para obtener el mayor rendimiento y que nuestro juego funcione aceptablemente en el mayor número de sistemas. Por lo general es imprescindible obtener rendimientos que no bajen de 30 frames por segundos (fps) en equipos medianos, para así evitar posibles saltos en el movimiento general del juego.

La modularidad, consistente en separar todo el código en pequeños programas, es también un buen método para organizarlo y estructurarlo mejor. Con este procedimiento resultará más fácil depurar errores (*bugs*) y, además, permite que varios programadores puedan trabajar en el mismo juego.

Otro concepto a tener en cuenta es procurar, lo máximo posible, el ahorro de memoria de sistema (RAM) y de vídeo, utilizando, por ejemplo, los tipos de variables adecuados o el tamaño de las texturas, controlando la creación y destrucción de entidades o borrando imágenes y audio en desuso.

Resumiendo, todas estas técnicas se aplicarán en la programación de "Zone of Fighters" de una forma u otra. Se van a utilizar unos métodos determinados, que llevarán a que el juego tenga una calidad aceptable.

(■) PREPARANDO Y ESTRUCTURANDO EL CÓDIGO

Vamos a basar la programación de "Zone of Fighters" en un sistema de modularidad. Para ello, dividiremos todo el código en pequeños programas (con extensión .bb), que luego serán incluidos desde el programa principal. Cada uno de estos programas se encargará de una tarea específica; de esta forma, podemos aislar los diferentes grupos de procedimientos dedicados a una labor en concreto.

```
MODULO "zone_of_fighters.bb"

Programa principal
    - Incluir módulos
    - Función Presentación
    - Función Definir modo gráfico
    - Función Cargar texturas y audio
    - Función Crear modelos
    - Función Crear emisores de partículas
      Bucle Principal
        - Inicializar número de luchadores
        - Función Menú
        - Si creamos un nuevo juego
          . Si el juego no es el primero
            Función Borrar entorno
            Función Borrar decorado
            Función Borrar enemigos
            Función Borrar mapa de colisiones
          . Función Crear entorno
          . Función Crear decorado
          . Función Crear OVNIS
          . Función Crear mapa de colisiones
        - Tocar música
        - Bucle de partida
            . Controlar FPS
            . Controlar tecla ESCAPE para salir
            . Función Controlar bionave
            . Función Actualizar entorno
            . Función Actualizar juego
            . Función Captura de la pantalla
            . Controlar la pausa
            . Actualizar mundo
            . Renderizar mundo
            . Función Indicadores de pantalla
            . Controlar visualización del panel ayuda. Tecla F5
          . Intercambiar los búferes
        - Repetir Bucle Principal hasta que se pulse ESCAPE
      Repetir Bucle Principal
Fin programa principal
```

1

Pseudocódigo del módulo principal "zone_of _fighters.bb".

A continuación se muestra una lista de los módulos y su cometido:

- **zone_of_fighters.bb:** Éste es el módulo principal del juego. Aquí se incluyen los demás módulos y es donde se encuentran el bucle principal y el bucle de cada partida.
- **definiciones.bb:** En este módulo se incluyen todas las definiciones de constantes, variables y estructuras que se utilizarán en el juego.
- **presentación.bb:** Aquí mostramos la presentación del juego. Pantallas del distribuidor, desarrollador y argumento.
- **menu.bb:** Este módulo contiene el menú principal y todos los submenús del juego.

- **funcarga.bb:** Aquí cargamos todas las texturas de los objetos del decorado, imágenes y todos los efectos de audio y voces.
- **funcpantaudio.bb:** Este módulo es el que contiene más funciones y se encarga de crear el entorno, los decorados, los modelos y el sistema de partículas. También controla los efectos especiales de audio y el modo gráfico.
- **decorado.bb:** Este módulo simplemente se encarga de colocar todo el decorado estático en el terreno.
- **jugador_principal.bb:** En este módulo, controlamos y actualizamos al jugador principal y todas sus características como el escudo, el camuflaje o los disparos.
- **jugadores_CPU.bb:** Aquí creamos y actualizamos a todos los OVNIS enemigos.
- **Fjuego.bb:** Este módulo contiene las funciones que manejan la cámara, el entorno, los indicadores, el mapa de colisiones y controla la evolución del juego.
- **enemigos.bb:** Con "enemigos" nos referimos a todas las plantas y animales que viven en el terrario. Así que este módulo los crea y controla. Además, también se encarga de los voladores o cubos kamicaces que pueblan la zona de combate.

Iremos por partes y explicaremos la estructura de cada módulo por separado mientras realizamos el pseudocódigo.

(■) MÓDULO "ZONE_OF_FIGHTERS.BB"

Éste es el programa principal. Realmente es pequeño ya que sólo incluye al bucle principal del juego, el cual contiene a su vez el bucle de cada partida. Al principio, incluimos todos los demás módulos por orden. Luego y después de la presentación, definimos el modo gráfico definitivo para el juego. Cargamos texturas y creamos modelos antes de entrar en el bucle principal. Una vez dentro de éste, llamamos al menú y, a continuación, creamos todos los elementos de la partida antes de introducirnos en el bucle de juego (Ver Fig. 1).

(■) MÓDULO "DEFINICIONES.BB"

No hay mucho que explicar aquí, sólo indicar los diferentes grupos de definiciones necesarios en el programa. No hay un orden específico a la hora de definir variables. Generalmente son las constantes las que encabezan la lista, así que empezaremos con ellas. Las utilizaremos para nombrar las entidades implicadas en el mapa de colisiones. Posteriormente, seguiremos con las variables que contienen las fuentes de letras y todas las matrices, variables y tipos que intervienen en el menú. A continuación, crearemos los *arrays* y estructuras necesarios para los bonos del juego, así como las utilizadas por los indicadores de pantalla.

Luego, definiremos todo lo necesario para el entorno del juego: zona de combate, variables de texturas del terreno, cielo, etc. Y para completar, las utilizadas por la cámara y las luces.

Seguimos con las variables globales utilizadas por el jugador principal y la estructura que define a los OVNIS. Para terminar con los jugadores, definimos la munición utilizada por ambos.

Le llega el turno al decorado, el cual necesita variables globales para el modelo y texturas y estructuras para controlarlos. De igual forma se define a los animales, plantas y voladores.

Terminamos las definiciones gráficas con las variables y tipos utilizadas para crear el sistema de partículas.

El módulo acaba con las variables referentes a todo el apartado sonoro del juego: música, efectos especiales de sonido (*sfx*) y voces (Ver Fig. 2).

```
MODULO "definiciones.bb"

- Definir constantes para el mapa de colisiones
- Definir fuentes de letras truetype que se utilizan para indicadores
- Definir valores iniciales para el entorno
- Matriz para los modos gráficos elegidos
- Matriz para las elecciones en el menú

- Estructura para los botones (opciones) del menú
- Matriz para el número de botones
- Matriz para los títulos de las opciones (gráfico con la lista de opciones)
- Variables de control del menú
- Matriz para las diferentes texturas de los bonos
- Matriz para el modelo de los bonos
- Estructura para controlar los bonos
- Definir variables para los indicadores de pantalla
- Definir variables para la cámara, luces y ambiente
- Definir variables para el control de la bionave (modelo, textura,
    pivote, sombra,coordenadas, velocidad, escudo, camuflaje,
    puntos, vida, munición y control para zurdos)
- Definir estructura para el control de los OVNIS (modelo, pivote,
    sombra, coordenadas, velocidad, munición, distancia objetivo,
    vida)
- Definir variables para munición de la bionave
- Estructura del disparo (sprite, transparencia, alcance, luz, tipo de disparo)
- Estructura de los agujeros y de las explosiones (sprite, luz)
- Definir variables para munición de los OVNIS
- Definir variables para los modelos del decorado, plantas y animales
- Estructuras para los modelos inanimados (modelo y coordenadas)
- Estructuras para los modelos animados (modelo, coordenadas, vida
    variables de la IA)
- Variables y Estructura para los voladores
- Variables para los 4 emisores de partículas
- Estructura de los 4 emisores (posición, duración, dirección y velocidad
    del viento, velocidad partículas, velocidad emisión, tamaño del
    emisor, peso y elasticidad de las partículas, sprite, vida,
    transparencia, ángulo de rotación y tamaño)
- Definir variables para el sonido (volúmenes, músicas, menú, explosiones,
    jugadores, plantas, animales y voces)
```

2

Pseudocódigo del módulo de definiciones "definiciones.bb".

(■) MÓDULO "PRESENTACIÓN.BB"

Se puede decir que es un módulo independiente del programa principal, ya que en él se define un modo gráfico y variables propias. Su sencilla labor es visualizar en pantalla los logos del distribuidor y del desarrollador, así como el argumento del juego (Ver Fig. 3).

(■) MÓDULO "MENU.BB"

El menú es algo más complejo. Consta de un fondo con movimiento donde se muestra la vista de una cámara rotando en círculo

```
MODULO "presentacion.bb"

- Definir modo gráfico
- Cargar imágenes (distribuidor, desarrollador,
    logo del juego, argumento)
- Cargar voz del argumento
- Mostrar Distribuidor
- Función Fade out Distribuidor
- Mostrar Desarrollador
- Función Fade out Desarrollador
- Reproducir voz con el argumento
- Mostrar Logo del juego
- Mostrar en scroll vertical imagen argumento
- Función Fade out Logo del juego
- Liberar imágenes de la memoria
```

3

Pseudocódigo del módulo de presentación "presentacion.bb".

```
MODULO "menu.bb"
Funcion menu
    - Definir fuente y establecer color tinta
    - Cargar imágenes logotipo y cursor
    - Inicializar valores de selección la primera vez
    - Crear botones y cargar sus imágenes
    - Asignar las coordenadas para las opciones (botones)
    - Tocar música del menú
    - Ocultar el terreno del juego y todo el decorado
    - Crear fondo del menú (luz ambiental, terreno, cielo, cámara)
      Bucle Principal del menú
        - Mantener la música sonando en todo momento
        - Función Actualizar el fondo del menú
      - Visualizar logotipo en la esquina
      - Controlar elección por el cursor del ratón
      - Según la opción elegida Función  panel de opciones
      - Actualizar cursor
    Repetir Bucle Principal del menú
    - Liberar memoria
    - Mostrar  terreno de juego y decorados si hay una partida en curso
Fin función menu
Funcion panel de opciones
    - Reproduce sonido clic
    Bucle de la función
    - Mantener la música sonando en todo momento
    - Función Actualizar el fondo del menú
    - Visualizar logotipo en la esquina
    - Si la opción es 0. Jugar
      . Petición al usuario del tipo de juego
        Si es tipo 1 (zona generada) Cargamos el terreno
        Si es tipo 2 (zona editada)
            Mostramos imagen de la zona y
            esperamos una elección.
            Cargar zona seleccionada
      . Inicializar variables de juego
      . Salir del menú
    - Si la opción es 1. Opciones del detalle gráfico
    - Si la opción es 2. Opciones de configuración del entorno
    - Si la opción es 3. Opciones volumen música
    - Si la opción es 4. Opciones volumen sfx
    - Si la opción es 5. Mostrar panel de controles
    - Si la opción es 6. Mostrar panel de créditos y finalizar programa
    - Controlar detección del cursor de todas las opciones
    - Actualizar cursor e intercambiar búferes
    Repetir hasta que se elija una opción
Fin función panel de opciones
Funcion actualizar fondo del menú
    - Mover cielo
    - Rotar luz del Sol (día/noche)
    - Rotar terreno
    - Actualizar y Dibujar mundo 3D
Fin función actualizar fondo del menú
```

4

Pseudocódigo del módulo del menú "menu.bb".

```
MODULO "funcpantaudio.bb"

Funcion definir modo gráfico
    - Definir modo estándar
    - Visualizar pane-
Fin funcion crear entorno
Funcion borrar entorno
    - Borrar luces, terreno, agua, camara y cielo
Fin funcion borrar entorno
Funcion borrar decorado
    - Borrar todos los edificios, árboles y rocas
    - Borrar todas plantas y animales
Fin funcion borrar decorado
Funcion borrar enemigos
    - Borrar todos los voladores
    - Borrar todos los OVNIS
    Fin funcion borrar enemigos
    Función definir modo gráfico
    Imprimir lista de los drivers gráficos del sistema
    - Esperar elección del driver
    - Visualizar panel
    - Imprimir lista de modos gráficos
    - Esperar elección del modo gráfico
    - Esperar confirmación de elección
    - Definir modo gráfico
    - Activar doble búfer
Fin función definir modo gráfico
Funcion crear entorno
    - Activar o desactivar antialisado
    - Crear Sol y luz ambiental
    - Cargar el terreno correspondiente y aplicar texturas
    - Crear cielo y visualizarlo si corresponde
    - Crear agua y crear reflejo si corresponde
    - Crear cámara y definir rango de visión y zoom
    - Crear niebla si corresponde
- Posicionar la cámara
Funcion ocultar decorado
    - Ocultar todos los edificios, árboles y rocas
    - Ocultar todas plantas y animales
    - Ocultar OVNIS y voladores
Fin funcion ocultar decorado
Funcion mostrar decorado
    - Mostrar todos los edificios, árboles y rocas
    - Mostrar todas plantas y animales
    - Mostrar OVNIS y voladores
Fin funcion mostrar decorado
Funcion crear modelos
    - Cargar y posicionar bionave, crear pivote y sombra
    - Cargar y texturizar OVNIS,
    - Crear y texturizar voladores,
    - Crear y texturizar bonos
    - Cargar, texturizar y posicionar edificios, árboles y rocas
    - Cargar, texturizar y posicionar plantas y animales
    - Crear los sprites de los disparos y agujeros de impactos
```

6A

Pseudocódigo del módulo de funciones gráficas y de audio (I) "funcpantaudio.bb".

```
Funcion crear decorado
    - Colocar el decorado según el tipo de juego elegido
      . Si es tipo de juego 1 (Generada)
          Llamar a las funciones para colocar los
          edificios, rocas, árboles, animales y plantas
          según el terreno elegido
      . Si es tipo de juego 2 (Zona editada)
          Leer datos del fichero de la zona editada
          Bucle lectura del fichero
           Leer registro
           Colocar modelo según el valor del registro objeto$
           Repetir lectura del fichero hasta el último registro
           Cerrar fichero
Fin funcion crear decorado
Funcion crear emisores de partículas
    - Crear emisor 1
    - Crear emisor 2
    - Crear emisor 3
    - Crear emisor 4
Fin funcion crear emisores de partículas
Funcion  emisor de partículas 1
    - Manejar y visualizar cada partícula
Fin funcion  emisor de partículas 1
Funcion  emisor de partículas 2
    - Manejar y visualizar cada partícula
Fin funcion  emisor de partículas 2
Funcion  emisor de partículas 3
    - Manejar y visualizar cada partícula
Fin funcion  emisor de partículas 3
Funcion  emisor de partículas 4
    - Manejar y visualizar cada partícula
Fin funcion  emisor de partículas 4
Funcion  actualizar emisores
    - Mantener "vivos" los emisores 1 y 3
Fin funcion  actualizar emisores
Funcion  fade out
    - Se realiza un fundido a negro colocando un sprite
          delante de la cámara
Fin funcion  fade out
Funcion  salvapantalla
    - Al pulsar return salvamos el búfer de pantalla
Fin funcion salvapantalla
Funcion  Existe Fichero
    - Retorna true si el fichero existe y false si no
Fin funcion  existe fichero
Funcion  Play sonido
    - Asignar el volumen a los efectos de sonido según
          distancia de la cámara al emisor del sonido
    - Asignar el panorámico de los efectos de sonido según
          los valores de proyección de la cámara
          con respecto al emisor del sonido que se encuentre
          a la vista.
Fin funcion  Play sonido
Funcion  Radar
    - Visualizar los OVNIS, voladores y bionave con
          diferentes puntos de colores según
          su posición en el terreno
Fin funcion  radar
```

6B

Pseudocódigo del módulo de funciones gráficas y de audio (II) "funcpantaudio.bb".

sobre un terreno de combate. Sobre todo esto se imprimen una serie de apartados por donde el usuario navega para seleccionar las diferentes opciones del juego. Primero, se crean todos los botones que servirán de opciones, luego el fondo y a continuación se entra en el bucle principal del menú, en el cual se actúa dependiendo de la decisión del usuario tomada con el ratón. Para terminar el módulo, se crean las funciones que controlan las diferentes opciones y la actualización del fondo (Ver Fig. 4).

```
MODULO "funcarga.bb"

Funcion cargar texturas y audio
    - Cargar fuente del juego
    - Establecer color de tinta y fuente
    - Cargar texturas de la urna y del cielo
    - Cargar imágenes de los disparos, indicadores de pantalla
        texturas de los bonos
    - Cargar imágenes de los paneles (victoria, muerte y controles)
    - Cargar textura de la bionave, los OVNIS y voladores
    - Cargar texturas de los edificios, árboles y rocas
    - Cargar texturas de las plantas estáticas
    - Cargar audio del menú

    - Cargar audio de la bionave, OVNIS y voladores
    - Cargar audio de las plantas y animales
    - Cargar audio de las voces
Fin función cargar texturas y audio
```

5

Pseudocódigo del módulo de carga "funcarga.bb".

(■) MÓDULO "FUNCARGA.BB"

En este módulo simplemente cargamos todas las imágenes utilizadas por el sistema de partículas, indicadores y desarrollo del juego. También cargamos las texturas utilizadas por los modelos inanimados. Y para finalizar, todos los sonidos y voces (Ver Fig. 5).

(■) MÓDULO "FUNCPANTAUDIO.BB"

Este módulo engloba todas las funciones que permiten definir el modo gráfico, crear y actualizar el entorno, decorado, enemigos y sistema de partículas. También incluye un sistema especial de reproducción del audio, así como funciones de control de archivos (Ver Fig. 6A y 6B).

(■) MÓDULO "DECORADO.BB"

La finalidad de este pequeño módulo es la de crear y colocar cada uno de los elementos inanimados sobre el terreno (edificios, árboles, rocas, etc.) (Ver Fig. 7).

```
MODULO "decorado.bb"

Funcion colocar decorado
    - Crear y colocar cada uno de los edificios, rocas y árboles
        sobre el terreno aleatoriamente, dentro de unos valores
        Prefijados
Fin funcion colocar decorado
```

7

Pseudocódigo del módulo de decorados "decorado.bb".

```
MODULO "jugador_principal.bb"
Funcion control jugador principal
    - Rotamos la bionave según el movimiento del ratón o de los cursores izq. y dcho
    - Control del cambio de vista
    - Control de la opción para zurdos
    - Control del disparo
      . Según el tipo de armamento seleccionado     crear disparo y actualizar munición
    - Control del escudo de la bionave
    - Control del camuflaje de la bionave
    - Control del cambio de armamento
    - Control de la velocidad de la bionave
    - Desplazar a la bionave y movemos la cámara
Fin funcion control jugador principal
Funcion actualizar jugador principal
    - Controlar la vida, el escudo y el camuflaje
    - Controlar la colisión con cualquier volador
    - Controlar la colisión con cada uno de los diferentes disparos de OVNIS
    - Controlar la colisión con los bonos
Fin funcion actualizar jugador principal
Funcion crear disparo
    - Crear un tipo u otro de disparo dependiendo del tipo de munición en uso
Fin funcion crear disparo
Funcion actualizar disparo
    - Destruir el disparo si ha llegado al fin de su alcance
    - Controlar la colisión del disparo con el terreno
      . Si son del tipo 3 y 4 creamos un agujero en el terreno
    - Controlar la colisión del disparo con los edificios
      . En caso de decorado interactivo dejamos un agujero por impacto
    - Controlar la colisión con los bonos. En caso de impacto destruirlos
    - Controlar la colisión con los árboles.  En caso de impacto moverlos
    - Controlar la colisión con los OVNIS
Fin funcion actualizar disparo
Funcion actualizar agujero
    - Eliminar mancha del agujero al cabo de un tiempo determinado
Fin funcion actualizar agujero
Funcion crear explosión
    - Crear una u otra explosión dependiendo del tipo de disparo  que la crea
Fin funcion crear explosión
Funcion actualizar explosión
    - Formar explosión y eliminarla al terminar
Fin funcion actualizar explosión
Funcion escudo bionave
    - Rotar gráfico del escudo
    - Mantenerlo activo hasta consumir el tiempo
Fin funcion escudo bionave
Funcion camuflaje bionave
    - Mantener la bionave transparente hasta consumir el tiempo de camuflaje
Fin funcion camuflaje bionave
Funcion ocultar bionave
    - Mantener la bionave oculta para los cambios de vista
Fin funcion ocultar bionave
Funcion mostrar bionave
    - Visualizar la bionave
Fin funcion mostrar bionave
Funcion muere jugador
    - Visualizar  gran explosión
    - Visualizar panel
    - Rotar la cámara alrededor de la bionave
    - Esperar respuesta del jugador
    - Inicializar variables de la bionave
Fin funcion muere jugador
```

8

Pseudocódigo del módulo de control del jugador principal "jugador_principal.bb".

```
Funcion crear jugadores CPU (OVNIS)
    - Crear tantos OVNIS como indique
        el número de jugadores elegidos
Fin funcion crear jugadores CPU
Funcion actualizar jugadores CPU
    Por cada OVNI:
    - Controlar muerte del OVNI y recompensar al jugador
        con vida, puntos y munición
    - Controlar colisión del OVNI con los disparos de
        la bionave y de ellos mismos.
        En caso de impacto, actualizar vida y
        cambiar inmediatamente de dirección
    - Controlar la colisión con los bonos
    - Decrementar contador interno de acciones
    - Buscar a quién disparar
        el 50 % posibilidades busca otros OVNIS
        el otro 50 % busca bionave y calculamos la
        distancia al objetivo
    - Si está a cierta distancia del objetivo disparar
        . Dependiendo de la distancia, disparar un tipo
            de arma u otra
    - Desplazar OVNI hacia el nuevo destino
    - Al llegar a cero el contador interno, fijar las
        nuevas coordenadas de destino donde
        se encuentre la bionave.
        Inicializar de nuevo contador interno
    - Al llegar a la mitad del tiempo del contador interno, fijar nuevas
        coordenadas de destino aleatorias
    - Posicionar al OVNI (sombra, partículas)
Fin funcion actualizar jugadores CPU
```

9

Pseudocódigo del módulo de control de los OVNIS "jugadores_CPU.bb".

```
MODULO "Fjuego.bb"

Función mueve cámara
    - Establecer coordenadas de la cámara para el seguimiento
        de la bionave, aprovechando la posición y pivote de éste
    - Posicionar la cámara
Fin función mueve cámara
Función cambiar cámara
    - Según el tipo de cámara seleccionada ajustar zoom, distancia y
        velocidad de seguimiento
Fin función cambiar cámara
Función actualizar entorno
    - Mover el cielo si existe
    - Si corresponde, Crear día y noche rotando la luz del sol y
        disminuyendo la cantidad de luz ambiental
    - Si es terreno tipo 1, generar partículas para los volcanes
Fin función actualizar entorno
Función actualizar juego
    - Mantener la reproducción de la música
    - Controlar el final de partida en caso de eliminar a todos los OVNIS
    - Función actualizar jugador principal
    - Función actualizar jugadores_CPU
    - Función actualizar lunys
    - Función actualizar dreecks
    - Función actualizar slunks
    - Función actualizar shaarks
    - Función crear voladores hasta un máximo determinado
    - Función actualizar voladores
    - Función actualizar emisores partículas
    - Función crear bonos hasta un máximo de 10 continuamente
    - Función actualizar bonos
    - Determinar los tiempos de carga para cada munición
    - Función actualizar disparos
    - Función actualizar explosiones
    - Función actualizar agujeros si procede
Fin función actualizar juego
Función indicadores
    - Visualizar gráficos de los indicadores de pantalla
    - Visualizar datos en los indicadores de pantalla
Fin función indicadores
Función mapa de colisiones
    - Definir tipos de colisiones
    - Crear mapa de colisiones
Fin función mapa de colisiones
Función crea bonos
    - Crear o nuevo bono y posicionarlo aleatoriamente en el terreno
Fin función crea bonos
Función actualiza bonos
    - Rotar cada bono
    - Crear un emisor de partículas en cada bono
    - Controlar impacto con un disparo
Fin función actualizar bonos
Función final partida
    - Reproducir música
    - Mostrar panel de victoria hasta que se pulse disparo
    - Borrar música
Fin función final partida
Función pausa
    - Detener el juego en un bucle hasta que se pulse la tecla "p"
Fin función pausa
```

10

Pseudocódigo del módulo de control del juego "Fjuego.bb".

(■) MÓDULO "JUGADOR_PRINCIPAL.BB"

Este módulo es fundamental y es el encargado de controlar todo lo referente a la bionave de combate controlada por el jugador. Detecta las teclas y el ratón y actualiza el comportamiento de la bionave en el desarrollo del juego. Además, controla todo lo referente a los disparos y sus consecuencias así como el escudo y sistema de camuflaje. Para finalizar, se completa el módulo con la función encargada de la muerte del jugador (Ver Fig. 8).

(■) MÓDULO "JUGADORES_CPU.BB"

Otro módulo importante, encargado de gestionar todo lo referente a los OVNIS que controla el ordenador. Está compuesto por una función que los crea y otra que controla su IA (Inteligencia Artificial) (Ver Fig. 9).

(■) MÓDULO "FJUEGO.BB"

El desarrollo del juego es controlado aquí, en este módulo. Se maneja el movimiento de la cámara y los cambios de vista. Además, es aquí donde se le da vida al entorno creando el día y la noche o los cambios climatológicos. Otra función de este importante módulo es la actualización del desarrollo de la partida, es decir, la actualización de todas las acciones realizadas por el jugador, OVNIS, plantas, animales, voladores y bonos. Además, se controla la interactividad del decorado con la acción.

También aquí se actualizan los indicadores de pantalla y se crea el mapa de colisiones (Ver Fig. 10).

(■) MÓDULO "ENEMIGOS.BB"

Los enemigos son colocados y actualizados en este módulo, es decir, las plantas, animales y cubos voladores (Ver Fig. 11).

El uso de la modularidad implica la definición de más variables globales, ya que la mayoría son compartidas por multitud de funciones. Sin embargo, no es de alarmar la cantidad de memoria que este procedimiento pueda necesitar. A cambio, obtenemos beneficios a la hora de depurar, optimizar o ampliar el código.

Otra observación es que se ha evitado totalmente el uso de etiquetas (".label") en el juego (sólo se ha utilizado una etiqueta en la presentación), rechazando así instrucciones de cambio de flujo como

```
MODULO "enemigos.bb"

Función colocar dreecks
    - Crear los dreecks especificados
    - Inicializar variables de ataque y contador de cambio de estado
    - Posicionar y animar cada dreecks
Fin función colocar dreecks
Función actualizar dreecks
    Por cada Dreecks:
    - Controlar la cantidad de vida restante
    - Controlar colisión con disparos
    - Atacar a la bionave si se encuentra cerca
        (cambiar animación del dreeck)
    - Animación de asechar cuando el contador llegue a la mitad
    - Animación en reposo cuando el contador llegue a 1/4
    - Animación de reposo cuando el contador llegue a 0
        inicializar contador y variables de ataque
Fin función actualizar dreecks
Función colocar lunys
    - Crear los lunys especificados
    - Inicializar variables de ataque y contador de cambio de estado
    - Posicionar y animar cada lunys
Fin función colocar lunys
Función actualizar lunys
    Por cada Luny:
    - Controlar la cantidad de vida restante
    - Controlar colisión con disparos
    - Animación de ataque cuando el contador llegue a la mitad
    - Animación lanzar esporas cuando el contador llegue a 1/4
    - Animación de reposo cuando el contador llegue a 0
        inicializar contador y variables de ataque
Fin función actualizar lunys
Función colocar slunks
    - Crear los slunks especificados
    - Inicializar variables de ataque y contador de cambio de estado
    - Posicionar y animar cada slunks
Fin función colocar slunks
```

11A

Pseudocódigo del módulo "enemigos.bb".

```
Función actualizar slunks
    Por cada Slunk:
    - Controlar la cantidad de vida restante
    - Controlar colisión con disparos
    - Animación salir, atacar y entrar en la tierra
        cuando el contador llegue a la mitad
    - Animación lanzar esporas cuando el contador llegue a 1/4
    - Reposo cuando el contador llegue a 0
        inicializar contador y variables de ataque
Fin función actualizar slunks
Función colocar shaarks
    - Crear los shaarks especificados
    - Inicializar variables de ataque y contador de cambio de estado
    - Posicionar cada shaark
Fin función colocar shaarks
Función actualizar shaarks
    Por cada Shaark:
    - Controlar la cantidad de vida restante
    - Controlar colisión con disparos
Fin función actualizar shaarks
Función crear volador
    - Crear un volador aleatorio de entre 4 diferentes
    - Posicionarlo aleatoriamente en el terreno
    - Inicializamos contador de estado
    - Asignamos coordenadas de destino (bionave)
    - Asignamos velocidad de desplazamiento
Fin función crear volador
Función actualizar voladores
    Por cada Volador:
    - Controlar la cantidad de vida restante
    - Controlar colisión con disparos
    - Controlar colisión con los bonos
        si colisiona con uno de ellos lo destruye
    - Desplazar volador hacia las coordenadas de destino
    - Fijar nuevas coordenadas de destino aleatorias cuando el contador
        llegue a la mitad
    - Fijar coordenadas de destino hacia la bionave cuando el contador
        llegue a 0
        Inicializar contador
        Inicializar nueva velocidad de desplazamiento
Fin función actualizar voladores
```

11B

Más pseudocódigo de "enemigos.bb".

"Goto" o "Gosub", principal enemigo de un programa estructurado.

Durante la programación del juego haremos diferentes versiones de algoritmos, modificaremos posibles bugs o incluso añadiremos cosas nuevas. Este método servirá para aprender aún más y poder añadir aspectos personales y diferentes al juego.

En el próximo número...

... una vez estructurado todo el trabajo, empezaremos visualizando el terreno de juego.

Grafismo 2D. Texturizado de la bionave de combate

Un buen modelo no sirve de nada sin buenas texturas que lo puedan cubrir y le den el aspecto deseado.

Para texturizar nuestra bionave de combate, utilizaremos una imagen que servirá de piel (*skin*), la cual cubrirá el modelo. En esta imagen estarán dibujadas todas las diferentes partes del modelo. Sin embargo, es necesario que cada una de estas porciones del dibujo esté ubicada de tal forma que al estirarse sobre el objeto coincida en los polígonos correctos. Para poder hacer práctica esta ubicación es necesario disponer de una plantilla del mapeado UV.

Antes de nada, necesitaremos crear el *UV Mapping* del modelo. Esto es necesario para que los vértices de los polígonos se adapten al dibujo.

Actualmente, existen multitud de aplicaciones para obtener este tipo de mapeados, las cuales utilizan sistemas de proyección para generar las coordenadas de la textura en 2D a partir de los puntos 3D del modelo. Otra función de estos programas es que permiten mover estas coordenadas en el plano 2D para conseguir un ajuste más perfecto. Una vez creado el UV Mapping, éste queda guardado con el modelo y servirá posteriormente de referencia para que Blitz3D o cualquier aplicación de modelado adapte la textura correctamente. El procedimiento tradicional de dibujar sobre una plantilla lo aplicaremos en el texturizado de otros modelos del juego; para la bionave, vamos a utilizar una aplicación que permite dibujar directamente sobre el modelo 3D, Deep Paint 3D.

Sin embargo, para poder pintar directamente sobre nuestra bionave de combate sin problemas, necesitaremos también un mapeado UV de ella. La versión 2.0 de este programa, incluida en nuestro CD-ROM, incorpora un programa para realizar este tipo de mapeados por medio de distintos tipos de proyecciones llamado *MercatorUV*.

DEFINICIÓN

▸ TEMPLATE

Una plantilla o template es simplemente una representación gráfica de un UV mapping (ver Fig. 1).

DEFINICIÓN

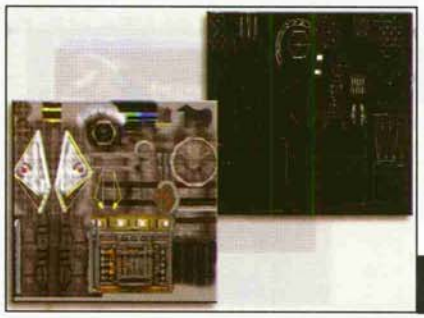
▸ UV MAPPING

UV Mapping o mapeado de coordenadas UV son posiciones colocadas en una imagen que se unen a puntos en un objeto 3D para ubicar una textura sobre él.

OPTIMIZAR EL MODELO

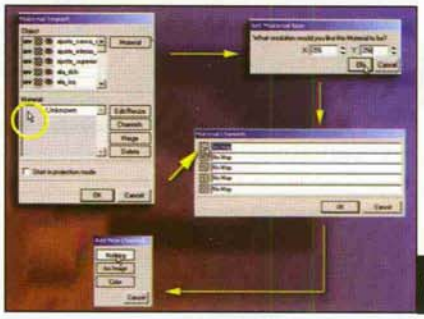
Antes de entrar en Deep Paint 3D, vamos a realizar una operación muy sencilla y útil que nos proporciona el programa Lithunwrap para optimizar nuestro modelo y así reducirlo de polígonos o vértices duplicados.

Dentro de Lithunwrap cargamos el modelo. Para optimizarlo elegimos la opción "Optimize models" situado en el menú "Tools". A continuación, con todas las opciones activadas pulsamos "Ok". Aparecerá una ventana de información, observamos cómo se ha conseguido reducir de polígonos la nave sin ninguna pérdida de calidad. Para terminar, salvamos de nuevo el modelo en formato .OBJ en la opción "Model" del menú "File".



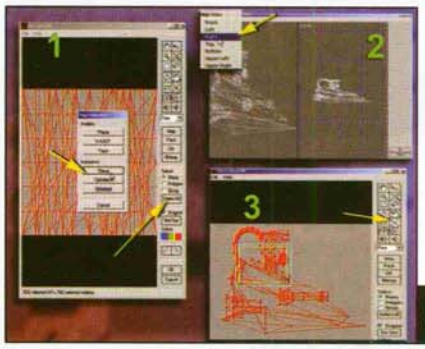
1

En la ilustración se muestra una imagen con toda la textura de la bionave y otra con la plantilla de despiece.



2

Los primeros cuatro pasos para empezar a trabajar con Deep Paint 3D.



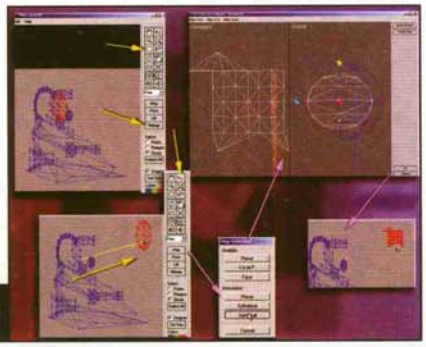
3

Los primeros pasos para empezar a trabajar con MercatorUV.

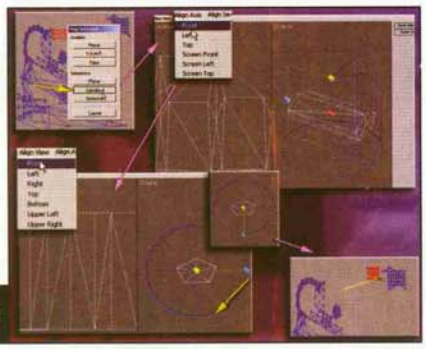
DEFINICIÓN

▸ TEXTURE MAPPING

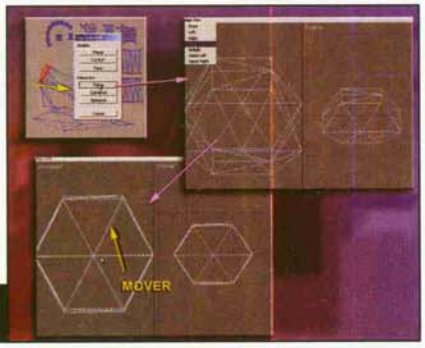
El mapeado de texturas (texture mapping) consiste en adaptar una imagen 2D (textura) a un polígono 3D.



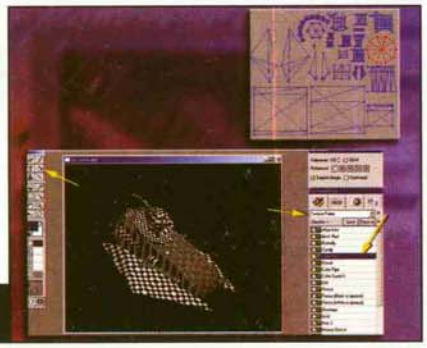
4 *Procedimiento para realizar el mapeado del casco de la bionave.*



5 *La cámara de la bionave necesita algunos ajustes para obtener el mapeado cilíndrico.*



6 *Con el ratón se puede mover directamente la pieza en la ventana "Unwrapped".*



7 *Una vez obtenido todo el UV Mapping, siempre es bueno realizar un chequeo.*

PRIMEROS PASOS CON DEEP PAINT 3D

Deep Paint 3D es una aplicación muy potente y cómoda de manejar. Su aspecto y la forma de utilizarlo se asemeja mucho al Adobe Photoshop. Iremos aprendiendo su uso a través de la práctica. A medida que trabajemos nuestros modelos descubriremos su funcionamiento fácilmente. Por lo tanto, vamos a proceder con el texturizado de la bionave. Empezamos cargando el modelo desde Deep Paint 3D en "File / Open". Aparecerá la ventana de diálogo "Material Import". Aquí, debemos crear un nuevo material para el modelo. Esto lo conseguimos pulsando al lado de "Unknow". En "Set Material Size" seleccionamos un tamaño para la textura de 256 x 256. Al pulsar "Ok" aparecerá "Material Channels", pulsamos en "C" para crear un canal de color y posteriormente en "Nothing" para tenerlo vacío (Fig. 2).

Ajustamos la vista pulsando dos veces en la herramienta lupa y rotamos en .

CREANDO EL UV MAPPING CON MERCATORUV

Como ya comentamos, debemos crear un mapeado de coordenadas UV antes de pintar sobre la bionave, para ello utilizaremos MercatorUV. Para entrar en esta aplicación pulsamos en el botón "Map" situado en la barra horizontal de herramientas. Una vez dentro del MercatorUV, debemos seleccionar todos los polígonos del modelo pulsando en "Select All". A continuación, crearemos una proyección lateral plana del modelo pulsando en el botón "Map" y eligiendo la opción "Planar" de "Interactive". Dentro de esta opción, elegimos la vista derecha "Right" en "Align View" (alinear vista) y "Ok". Con la herramienta "Scale" (escalar) reducimos la plantilla hasta que quede dentro del fondo gris (Fig. 3).

Debemos separar, manualmente, los grupos de polígonos que forman cada parte del modelo para realizar un mapeado individual. Para seleccionar cada grupo hay que activar la opción "Group" en "Select", luego con la herramienta de selección (M) elegimos un poco del casco. Al tener activada la opción "Group", MercatorUV automáticamente selecciona todo el casco, el cual se volverá de color rojo. Con la herramienta mover (T), lo separamos del modelo hacia cualquier parte del fondo gris. El siguiente paso será crear el mapeado del casco. Seleccionamos el mapeado interactivo esférico "Map / Interactive / Spherical". Se abre una ventana, en la que podemos ver cómo el casco se ha desenrollado (unwrapped) (Fig. 4).

Para el siguiente grupo, la cámara situada sobre el casco, elegiremos un mapeado cilíndrico "Map / Interactive / Cylindrical". Elegimos la vista frontal ("Align View / Front") y luego el alineamiento frontal de los ejes en "Align Axis / Front". A continuación, movemos la esfera azul (eje horizontal) de la ventana "original" hacia la parte inferior y pulsamos "Ok". Terminamos colocando el grupo junto al del casco (Fig. 5).

A continuación, se muestra una lista con el tipo de mapeado aplicado al resto de los grupos del modelo:

- **Conector al casco:** Map / Interactive Spherical ➡ Ok
- **Cuello y ajuste del cuello:** Map / Interactive Planar / Align View Top ➡ Ok
- **Tubo:** Map / Interactive Planar / Align View Right ➡ Ok
- **Ajuste tubo:** Map / Interactive Cylindrical ➡ Ajustes con las esferas amarilla y azul
- **Filtros, boquilla, escape y cubre escape:** Map / Interactive Cylindrical ➡ Ok

NOTA

En Deep Paint 3D podemos tener hasta cinco canales de efectos simultáneos distintos por cada material. "C" Color, "B" Bump Mapping (profundidad), "G" Glow (resplandor), "S" Shine (brillo) y "O" Opacity (opacidad).

■ **Codificador:** Map / Interactive Planar / Align View Top ➡ Movemos el objeto en la ventana "Unwrapped" hasta verlo desde arriba (Fig. 6).

■ **Extremo cañón y cañón:** Map / Interactive Planar / Align View Left ➡ Ok

■ **Base del cañón, alas y alerones:** Map / Interactive Planar / Align View Top ➡ Ok

■ **Parte superior del casco:** Map / Interactive Planar / Align View Top ➡ Ok

■ **Parte inferior del casco y tapa del motor:** Map / Interactive Planar / Align View Bottom ➡ Ok.

Estos dos últimos debemos reducirlos para que encajen dentro del fondo gris, con la herramienta de escalado.

Para finalizar, ajustamos todos los grupos con mucho cuidado dentro del fondo gris.

TEXTURIZANDO CADA GRUPO

Una vez creado el UV Mapping podemos pintar sobre la bionave directamente. Antes, es posible realizar un chequeo del mapeado, rellenando mediante la herramienta (K) con la textura "Checker+" de la lista "Texture Paints" ("Preset" F5) en el panel de comandos ("Command Panel"). Una vez comprobado, borramos la acción de relleno con "Undo" (CTRL+Z) (Fig. 7).

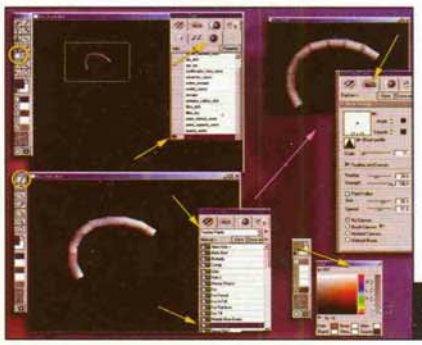
Vamos a comenzar dibujando el casco del protagonista, empezando por el tubo. Para ello, ocultaremos todos los demás elementos. En el panel de comandos ("Panel Command") pulsamos en "Element" (F7) y luego en "Objects". Aparecerá una lista con todos los grupos que forman la bionave. Ocultamos todos los grupos que no nos interesan pulsando en el "ojo" de cada capa. (Fig 069, 071). Una vez aislado el tubo, lo pintaremos con aspecto metálico. Elegimos la textura "Mercury" en "Preset / Texture Paints" y seleccionamos el tubo con la herramienta de selección (M). Hacemos un zoom hacia la pieza y con la herramienta de pincel, pintamos sobre la selección. Para perfilar las uniones entre segmentos, elegimos un pincel estándar y un color gris oscuro (Fig. 8).

Seguimos con los soportes del tubo. Borramos selección con CTRL+D, ocultamos el tubo y mostramos los soportes. Elegimos la textura "Metal Plate 7+" y pulsamos en "Brush and Paint Setting" (F6) para modificar la textura. Para obtener todas las opciones de edición, debemos pulsar sobre el símbolo "+" en "Texture Behavior" (comportamiento de la textura) y en "Advanced Behavior". Vamos a cambiar el tamaño de la textura moviendo totalmente hacia la izquierda el deslizador "ScaleX". Elegimos el pincel y pintamos los soportes. Para el conector del casco vamos a elegir la textura "Si-fi 3 +". En "Advanced Behavior"desplazamos el deslizador de tono "Hue"hacia -248 para cambiar el color y pintamos (Fig. 9).

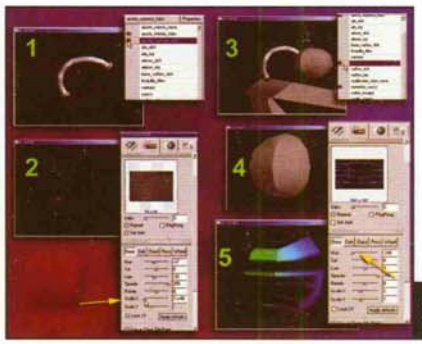
Para pintar el casco, lo aislamos y seleccionamos un color azul oscuro para luego pintar con el aerógrafo "AirBrush Sharp" en "Standard Tool" en el panel de comandos. Para dar la sensación de cristal, dibujaremos un brillo en el casco con "Airbrush Medium". Para realizar un pintado con más detalle podemos pasar al modo proyección pulsando en el icono (Fig. 10).

Siguiendo con la boquilla, la pintamos con la textura "Shift 3+" y la modificamos en tamaño, tono (Hue) y saturación (Saturation). Para los filtros, primero los pintamos de gris oscuro con una herramienta estándar ("Simple Flat") y luego con la textura "Alien Wire+" modificada en tamaño, color y luminosidad (Fig. 11).

Para el cuello y soporte del casco, elegimos la textura "Old metal 1". Una vez rellenas las piezas, vamos a proceder a pintar algunos detalles como radios y tornillos. Para dibujar los radios, mezclamos la visualización de la textura con los polígonos en wireframe con la opción "Display Wireframe", en la ventana emergente que aparece al pulsar el botón derecho del ratón sobre la vista. A continuación, seleccionamos el pincel "Gouache Bristle+"



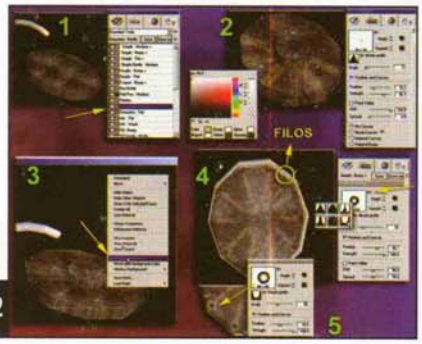
Procedimientos para el pintado del tubo.



Cómo pintar los soportes del tubo y el conector del casco.

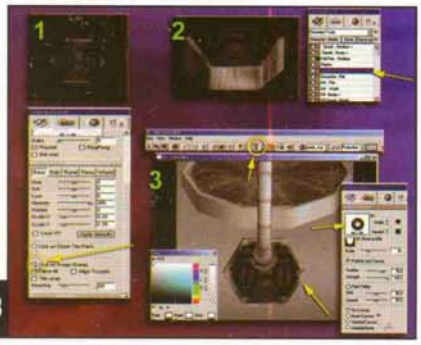
Procedimientos para pintar el cristal del casco.

Pasos para el pintado de la boquilla y los filtros.



12

Un trabajo especial de detalle requiere el cuello y su soporte.



13

Con la opción "Use as image stamp" colocaremos la textura como si se estampa un sello sobre el papel.

14

El modo de proyección nos ayudará a realizar un pintado con más detalle.

15

Procedimientos para el acabado de la estructura de la bionave.

y pintamos las líneas. Los tornillos los dibujamos en el modo de proyección. Tenemos que cambiar el tipo de pincel en "Brush Profile" y elegir la última opción, ajustamos la escala ("Scale") a 32, "Feather" a 66.7, Strength a 100.0 y pintamos los tornillos (Fig. 12).

Para rematar los filos y cantos de estas piezas, elegimos "Oil Bump" en "Standard Tools" y un color blanco. Para el codificador, elegimos la textura "Si-Fi 5". En "Advanced Behavior" activamos las opciones "Use as Image Stamp" y "Allow Tilt" para colocar la textura sin tilear. Ajustamos el tamaño "ScaleX y "ScaleY" a 0.39 y damos un sólo toque con el pincel en el centro de la pieza. Con "Gouache Bristle +" en "Standard Tools" perfilamos los bordes y el canto y, para los tornillos, cambiamos al modo proyección y pintamos con el mismo "Brush" que utilizamos en los tornillos anteriores. (Fig. 13).

Seguimos con la cámara. La aislamos del resto de piezas y seleccionamos su parte superior con la herramienta de selección (M) con "Face Selection Tools y "Feather" de 1. Pintamos con el pincel "Pastel Chalk" y un color casi negro. Con este mismo pincel dibujamos el frontal y la línea central de la parte superior. A esta línea le aplicamos el efecto "Emboss From Layer Below+" de la lista "Imagen Processing" con la herramienta de línea en el modo *proyección* (Fig. 14).

Para la parte superior de la nave, elegiremos la textura "Si-Fi 7 +", le reducimos la saturación de color, activamos de nuevo "Use as Image Stamp" y pinchamos en la pieza. Observamos cómo la textura cubre por completo la pieza. Si se coloca un poco desplazada podemos pinchar, un poco desplazado cada vez, en el objeto hasta que quede totalmente centrada. Empezamos detallando esta pieza, cambiando la tonalidad de la textura en aquellas partes que hagan contacto con otras piezas. Por ejemplo, el ajuste del cuello situado sobre la nave. Elegimos la opción "Darken" de la lista "Imagen Processing" y bordeamos con la herramienta pincel los contactos entre las piezas. Para la parte inferior, utilizaremos la textura "Si-Fi 5+" modificando "Sat" a -85 y "Lum" a 54 y de nuevo activamos "Use as Image Stamp". Para pintar los bordes laterales de las dos partes de la nave simularemos un material plateado con el pincel "Oil Bump+" y un color blanco (Fig. 15).

Pasamos a dibujar las alas y alerones, utilizaremos la textura "Si-Fi 5" con las siguientes modificaciones: "Hue" 0, "Sat" -255, "Lum" 55, "ScaleX 0.18,y "ScaleY" 1.0. Luego activamos "Use as Clone Tile Paint" en "Advanced Behavior" para hacer copias de la textura sobre la pieza con un ajuste tileado. Detallamos las alas con franjas naranjas en sus extremos que podemos realizar con la herramienta de línea. Aclaramos la textura del ala mediante un efecto "Lighten +" de la lista de "Imagen Processing" y pintando con un pincel estándar. La parte trasera de los alerones la pintamos con un color brillante y aplicamos un poco de "Glow" en "Imagen Processing" para simular luces encendidas (Fig. 16).

Para la tapa del motor aplicamos de nuevo la textura"Si-Fi 5 +" con los siguientes datos de modificación: "ScaleX" 0.23 y "ScaleY" 0.49. Y colocamos la textura con la misma opción "Use as Clone Tile Paint". En el escape y cubre escape utilizamos la misma textura "Si-Fi 5+". La base de los cañones la cubrimos de un color oscuro y luego pintamos unas franjas anaranjadas. Para terminar el texturizado de nuestra nave quedan los cañones. Los pintamos con un color gris oscuro, luego con un proceso de oscurecimiento ("Darken +" en "Imagen Processing") dibujamos algunas líneas y manchas para dar el aspecto de que está manchado del humo de los disparos.

En el próximo número...

... añadiremos movimiento a nuestra bionave con CharacterFX.

Creando el sonido de los disparos, explosiones y colisiones

Ya estamos preparados para realizar los sonidos de "Zone of Fighters". Algunos de ellos partirán de otros sonidos procedentes de librerías de libre uso, los cuales modificaremos a nuestro gusto para obtener un nuevo sonido.

Generalmente, éste suele ser el procedimiento utilizado por la mayoría de las desarrolladoras, sobre todo si se trata de sonidos reales como ruidos específicos, sonidos de la naturaleza, etc. Sin embargo, también crearemos efectos sonoros desde cero, a partir de grabaciones desde micrófono o de partes extraídas de un CD. La idea básica que pretendemos es que el lector sepa sacar provecho de todas las posibilidades del editor de audio para crear sonidos nuevos a partir de otros. Antes de empezar debemos tener apuntado en papel una lista con los nombres que asignaremos a cada uno de los ficheros de audio que creemos para no despistarnos. "Zone of Fighters" utiliza cerca de 40 sonidos diferentes, así que explicaremos aquí los más significativos, suficientes para comprender los procedimientos para la creación de audio.

Todos los sonidos que fabriquemos los almacenaremos en el directorio "C:\juego_ZOF\Audio\". Toda esta organización de ficheros que estamos realizando hasta ahora nos servirá al final para crear la instalación del juego.

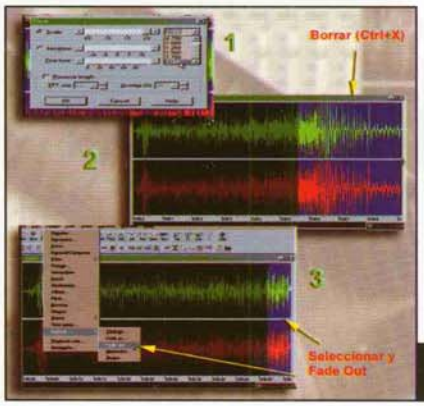
CREANDO LOS DISPAROS

Como sabemos, nuestra bionave de combate puede disparar distintos tipos de armas. Cada una sonará de una forma diferente al ser disparada, así que tendremos que realizar cuatro sonidos distintos. Además, los ovnis enemigos poseen tipos de armas análogas, por lo que reproducirán los mismos sonidos que la bionave. Todos los sonidos de los disparos los vamos a fabricar a partir de otros obtenidos de una librería. Solamente explicaremos los que necesiten operaciones diferentes.

MUNICIÓN DE BAJO CALIBRE

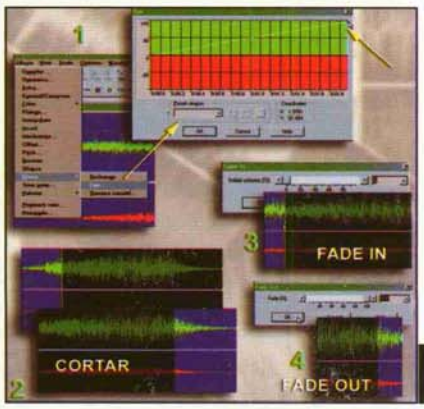
Ejecutad el programa Goldwave y cargar el sonido "sonido1.wav" que se encuentra en el directorio "Extras" del CD que adjuntamos. En la reproducción podemos apreciar que se trata del sonido de una puerta automática. En primer lugar debemos establecer qué calidad vamos a utilizar en el juego. Para este tipo de efectos utilizaremos sonidos mono a 11025 Hz y 8 bits de calidad. Debemos recordar que antes de convertir a esta calidad es conveniente realizar todas las operaciones de edición. Así que empezaremos normalizando el volumen con la opción

"Maximize" en "Effects \ Volume". Subimos a 1.5 en "New maximum". Al ser un disparo que se ejecutará rápidamente, necesitaremos un sonido corto y de ejecución rápida también. Así que tendremos que aumentar el tono aumentando la escala del "Pitch" también un punto más en "Pitch \ Effects" y subir a 2.000 en "Scale". Aun así comprobamos que sigue siendo algo largo, por lo tanto tendremos que cortarlo un poco. Para ir probando antes de cortar, iremos situando la línea de posición final y pulsamos "Play" para reproducir la selección. Una vez que hemos elegido la porción que nos interesa situamos las líneas de posición englobando el trozo que queremos borrar y pul-



Gracias a la opción fade, lograremos un buen acabado.

Si resampleamos de nuevo, obtendremos la calidad deseada.



Crearemos el sonido de los misiles.

4

Crearemos las bombas de retardo.

NOTA

▸ SONIDOS DE COLISIONES

Para crear el sonido de un impacto metálico similar al que puedan sufrir la bionave o los ovnis, se podrían utilizar efectos de reverberación o mecanizado a un sonido de golpe seco (sonido4.wav del CD) Sin embargo, la mejor forma es la tradicional, es decir, grabar a través del micrófono un golpe a algo metálico como un utensilio de cocina o un trozo de chapa. También podemos utilizar el sonido de los metales de una percusión de un teclado externo.
En resumen, para obtener el sonido deseado es sólo cuestión de tener clara la idea de qué es realmente lo que se quiere. Los sonidos procedentes de librerías de libre uso constituyen una buena base para experimentar con todas las posibilidades del editor de sonido. Hemos podido comprobar cómo, modificándolos adecuadamente, se pueden obtener sonidos realmente diferentes y acordes con nuestras necesidades.

samos CTRL + X. Volvemos a situar las líneas de posición englobando todo el sonido o CTRL + A para seleccionarlo completo y probamos el resultado. Para lograr un buen acabado, haremos un "Fade out" al final para "matar" el sonido. Seleccionamos un trozo del final y aplicamos el fade en "Effects \ Volume \ Fade out" con valor 100 (Fig. 1).

Una vez hayamos terminado, debemos pasarlo a la calidad que vayamos a utilizar. Para ello salvamos el sonido en "Save as" y en la opción "File Attributes" elegimos "8 bits, mono, unsigned", damos un nombre al archivo (nosotros hemos utilizado una "S" de "sonido" delante para identificar al fichero), "Sdisparo1" y guardamos en el directorio establecido "C:\juego_ZOF\Audio\". Al salvar el programa nos preguntará si queremos actualizar el sonido que estamos editando por el nuevo sonido cambiado de formato. Al decirle que "Sí" el sonido que estábamos editando cambia. En la barra de inferior de información observamos que todavía sigue a calidad 22025 Hz. Para pasarlo a la calidad deseada, es decir 11025 Hz debemos "Resamplearlo" de nuevo con la opción "Effects \ Resample" y eligiendo el valor 11025 (Fig. 2).

Para salvar es suficiente con pulsar CTRL + S.

MISILES

Para los misiles partiremos de un sonido de avión. Cargamos "sonido2.wav" del CD. Aquí nos encontramos con un problemilla. Observamos que está en estéreo y además el panorámico cambia de izquierda a derecha. No debe de ser un problema, porque al pasar a mono unificamos los dos canales, pero seguramente perderemos algo del efecto que buscamos. Así que, para evitar cualquier contratiempo, vamos a modificar el panorámico de nuevo. Elegimos la opción "Effects \ Stereo \ Pan" y subimos al máximo el punto final de la gráfica. A continuación, normalizamos y borramos los extremos como se muestra en la figura 3

Realizamos un "Fade in" (a 0) al principio y un "Fade out" (a 100) al final y salvamos con la calidad 8 bits, mono unsigned con el nombre "SDisparo2".

BOMBAS DE RETARDO

Para construir este sonido cargamos "sonido3.wav" del CD. Se trata del sonido de un disparo cualquiera, pero nosotros lo cambiaremos hasta obtener el que deseamos. Primero vamos a ecualizar para obtener más graves. Entramos en la opción "Effects \ Filtres \ Equalizer" y subimos en 24 Db la frecuencia de 60 (dos veces 12 Db) y bajamos en 12 la de 6000 Hz. Luego modificamos la escala del tono a 0.75. Continuamos dándole la vuelta con "Reverse" y añadiendo un silencio al principio, según las marcas de selección y eligiendo "Effects \ Silence". (Fig. 4).

CREANDO LAS EXPLOSIONES

Realmente, las explosiones se pueden crear de muchas formas, desde grabar una onomatopeya a través del micrófono, modificando un sonido ya existente o a partir de un ruido generado como el viento. Nosotros vamos a crear una explosión general, la cual nos servirá para aprender la técnica para cualquier tipo de explosión. Partiremos del sonido anterior "sonido2.wav". Una vez cargado, lo convertimos a mono como explicamos anteriormente (podemos utilizar cualquier nombre de archivo, por ejemplo, "explosion". Normalizamos y aislamos la parte central eliminando los extremos. A continuación, simplemente bajamos el tono en 0.5. Para terminar tendremos que ajustar el final con un "Fade out".

En el próximo número...

... explicaremos las pautas a seguir para crear sonidos de ambiente y el tratamiento adecuado de la voz para conseguir la "voz en off" de nuestro juego.

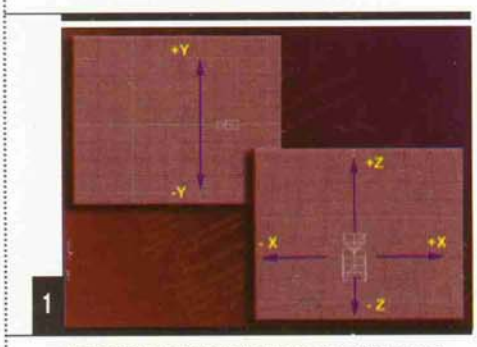
Manejo de entidades

Para Blitz3D, todo lo que se desenvuelve dentro de un mundo 3D se denomina "Entidad". Los sprites que estudiamos en la entrega anterior forman parte de estas entidades.

Pero además, hay otros elementos que pertenecen a este concepto, como son: cámaras, luces, objetos 3D, planos, terrenos y pivotes. Cada una de estas entidades tiene una labor diferente dentro de un programa, sin embargo, todas poseen algo en común: una posición, una rotación y un tamaño determinado. Esto significa que pueden ser movidas, rotadas o escaladas. Además, todas las entidades pueden tener una entidad "madre" a la que estarán unidas. Si la "madre" se mueve, rota o escala, la entidad "hijo" también lo hará.

NOTA

Las entidades se posicionan utilizando el sistema de coordenadas X, Y, Z. El centro del mundo 3D es la posición 0, 0, 0, que es donde se colocan todas las entidades por defecto, incluida la cámara.



1

Las entidades vistas desde la cámara se mueven en los ejes x, y y z.

No abordaremos, de momento, todas las funciones de manejo de entidades en este número, pero sí las más importantes. De todas formas, durante el estudio de las distintas entidades, encontraremos nuevas funciones para aprender.

MOVIENDO ENTIDADES

Encontramos algunas funciones para desplazar, rotar y escalar las entidades. La primera función que estudiaremos será cómo posicionar una entidad en el mundo 3D.

```
PositionEntity entidad,
coordenada X#, Y#, Z#
```

Con esta función, la entidad también se puede desplazar cambiando algunas de las coordenadas, pero lo hará siempre de una manera relativa con respecto a la posición y orientación de la cámara.

Para mover una entidad dependiendo de su posición y orientación utilizaremos:

```
MoveEntity entidad,
coordenada X#, Y#, Z#
```

Si por el contrario sólo queremos que dependa de su posición utilizaremos:

```
TranslateEntity entidad,
coordenada X#, Y#, Z#
```

Para comprender mejor estas funciones, vamos a adquirir una entidad de tipo *mesh*, es decir, un objeto 3D, por ejemplo, un cilindro, el cual utilizaremos en los ejemplos. En "ejemplo8_1.bb" se puede ver el funcionamiento de estas tres funciones realizando las mismas operaciones (Fig. 2).

2

Hay varias funciones para mover entidades.

Tenemos también dos formas de rotar una entidad: *RotateEntity* y *TurnEntity*:

```
RotateEntity entidad, pitch#,
yaw#, roll#
TurnEntity entidad, pitch#,
yaw#, roll#
Pitch es referente al eje X,
yaw al Y y roll al eje Z.
Repeat
...
  RotateEntity cilindro1,30,0,0
  TurnEntity cilindro2,30,0,0
...
Forever
```

En este bucle, se rota el "cilindro1" sólo 30 grados en su eje X (orientación absoluta), mientras que el "cilindro2" rotará conti-

NOTA

Es importante recordar siempre que si la cámara se mueve de derecha a izquierda lo estará haciendo en el eje X (dcha +X / izq –X), si sube o baja, en el eje Y (subir +Y / bajar –Y) y si avanza o retrocede, en el eje Z (avanzar +Z / retroceder –Z). Lo mismo ocurre con las entidades vistas desde la cámara (Ver Fig. 1).

3

Los dos cilindros tienen distintos grados de rotación.

nuamente de 30 en 30 grados en su eje X (posición relativa a su orientación) (Ver ejemplo8_2.bb) (Fig. 3).

Con PointEntity, podemos colocar una entidad en el sitio de otra. Su estructura es la siguiente:

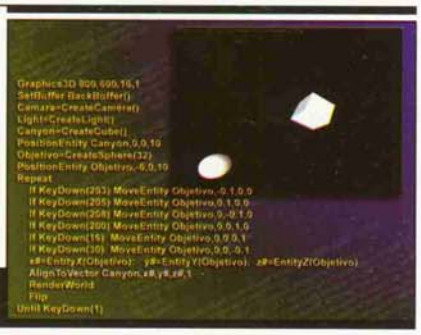
```
PointEntity entidad,
entidad de destino
```

También podemos alinear los ejes de una entidad a un vector determinado, es decir, con esta función podemos hacer que una entidad gire en busca de otra. Un buen ejemplo de este tipo de funcionamiento es el seguimiento que un cañón situado en una

DEFINICIÓN

▸ VECTOR Y NORMAL

En un espacio 3D, un *vector* es un segmento de recta colocado en cualquier punto y con una dirección determinada. Una *normal* es un vector perpendicular a cualquier polígono.



4

Ejemplo de una entidad que gira en busca de otra.

torreta hace al objetivo (Ver ejemplo8_3.bb) (Fig. 4).

Utilizaremos la función:

```
AlignToVector entidad, vector
X#, vector Y#, vector Z#, eje
[,proporción]
```

El "eje" de la "entidad" se alineará al "vector X#, vector Y#, vector Z#" con una transición de alineamiento según "proporción" (para Blitz3D superior a la versión 1.66).

CONTROLANDO ENTIDADES: CONTROL BÁSICO

En toda manipulación de entidades es fundamental disponer de funciones para su control. Podemos empezar con las que controla su visibilidad. Para ocultar una entidad y que además no pueda ser detectada utilizaremos *HideEntity entidad*. Si por el contrario queremos detectarla pero no verla, la mejor forma es volverla transparente poniendo a 0 su canal alfa con *EntityAlpha entidad,0*.

Para volver a mostrarla utilizamos *ShowEntity entidad* o *EntityAlpha entidad,1*.

Una función muy interesante y utilizada es *CopyEntity entidad [,pariente]*. Permite copiar una entidad en otra, es decir, crear un clon. Muy útil en la creación de tipos de entidades.

Si en un programa dejamos de utilizar una entidad es fundamental borrarla para salvar memoria, para ello utilizaremos la función *FreeEntity entidad* (Ver ejemplo8_4.bb) (Fig. 5).

CONTROLANDO ENTIDADES: EFECTOS ESPECIALES Y OTROS

A las entidades se les pueden aplicar algunos tipos de efectos como color, transparencias, brillo especular, mezclas, vértices coloreados, etc.

Por ejemplo, *EntityColor entidad, rojo#, verde#, azul#* proporciona un color a una entidad.

Para conseguir un brillo especular en un objeto, es decir, que

5

Con FreeEntity, borraremos una entidad.

se le iluminen ciertas áreas cuando recibe una luz directa, utilizaremos la función *EntityShininess entidad, brillo#*. El rango de "brillo" va desde 0 hasta 1. Así que un brillo medio sería 0.5.

También podemos realizar efectos de mezcla entre varias texturas aplicadas a un objeto con la función *EntityBlend entidad, tipo de mezcla*.

En un próximo número estudiaremos con más detalle este tipo de función cuando aprendamos el poder de Blitz3D para texturizar objetos. Pero podemos anticipar ya la función que permite realizar esta acción mágica, se trata de *EntityTexture entidad, textura [,fotograma][,indice]*.

Aquí, "textura" es la variable que contiene la textura cargada de disco y "fotograma" (frame) especifica qué fotograma de la textura animada se utilizará. Blitz3D permite mezclar varias texturas en el mismo objeto -este proceso se denomina "Multitexturing", por ejemplo, lo utilizamos en el texturizado del terreno de combate. De tal forma que "indice" indica el número de la textura asignada al objeto (de 0 – 7).

Pero hay una función especialmente preparada para ofrecer algunos efectos adicionales a una

NOTA

Es importante saber que las funciones de visibilidad aplicadas a una entidad afectarán también a sus entidades hijos si las tuviera.

entidad, nos referimos a *EntityFX entidad, efecto*. Hay varios tipos de efectos, desde aplicar brillo a todo el objeto hasta mostrar, en modo wireframe (alámbrico), todos los polígonos.

Para obtener la madre de una entidad se utiliza la función *GetParent Entidad* y para unirla a otra utilizaremos *EntityParent entidad hijo, entidad madre*.

EL ESTADO DE LAS ENTIDADES. OBTENIENDO INFORMACIÓN

En este apartado encontramos funciones muy potentes que nos permitirán obtener información muy útil de cada entidad. Vamos a comenzar con unas funciones básicas, las cuales nos proporcionarán en cualquier momento la situación de una entidad en el mundo 3D a través de sus coordenadas.

```
EntityX# (entidad) EntityY#
(entidad) EntityZ# (entidad)
...
PositionEntity cubo,0,3.5,-10
PosX#=EntityX(cubo):
PosY#=EntityY(cubo):
PosZ#=EntityZ(cubo)
Text 10,10,"La posición X del
cubo es: " + PosX ;← PosX= 0
Text 10,30,"La posición Y del
cubo es: " + PosY ; ← PosY=3.5
Text 10,60,"La posición Z del
cubo es: " + PosZ ; ← PosZ= -10
...
```

De igual forma podemos obtener los ángulos X, Y y Z de una entidad a través de las funciones:

```
EntityPitch# (entidad)
➡ ángulo de rotación X
EntityYaw# (entidad)
➡ ángulo de rotación Y
EntityRoll# (entidad)
➡ ángulo de rotación Z
```

Podemos encontrar en Blitz3D cuatro funciones realmente interesantes para implementarlas en comportamientos (IA) de entidades. Nos referimos a: *EntityVisible, EntityDistance, EntityPick y LinePick*.

```
EntityVisible ( entidad
fuente, entidad destino )
```

Esta función retornará "verdadero (True o 1)" si la "entidad fuente" puede ver a la "entidad destino".

```
EntityDistance# (entidad
fuente, entidad destino)
```

Con esta función obtendremos la distancia existente entre una entidad ("entidad fuente") y otra ("entidad destino").

```
EntityPick (entidad fuente,
rango#)
```

Esta función devolverá la entidad más cercana a "entidad fuente" en un radio de acción definido por "rango". Pero antes de poder aplicar esta función, es necesario que la entidad que se detecte tenga que ser "pickeable"o escogida, con la función *EntityPickMode*, la cual detallamos más adelante.

```
LinePick ( entidad fuente,
X#, Y#, Z#, DX#, DY#, DZ#
[,radio de la línea] )
```

Es una función muy parecida a la anterior, la diferencia es que ésta retornará la primera entidad que se encuentre en una línea desde las coordenadas X, Y y Z hasta X + DX, Y + DY y Z + DZ.

Para completar nuestra búsqueda de información sobre entidades encontramos dos funciones que nos dirán si una entidad está animándose y cuál es la duración de la animación:

```
EntityAnimating (entidad)
```

➡ Devuelve verdadero o falso si la animación de la entidad está activa o no.

```
EntityAnimTime# (entidad)
```

➡ Devuelve un valor numérico indicando la duración de la animación.

Para terminar, podemos transformar un punto, vector o normal de una entidad a otra con las funciones *TFormPoint, TFormVector* y *TFormNormal*. Estas funciones son muy interesantes, por ejemplo, para saber el lugar exacto del terreno donde ha caído una bomba.

Las coordenadas transformadas se pasan a las funciones *TFormedX, TFormedY* y *TFormedZ*.

```
TFormPoint X#, Y#, Z#, entidad
fuente, entidad destino.
```

Si en lugar de la entidad fuente o destino pusiéramos el valor 0, se usaría el espacio global para el cálculo.

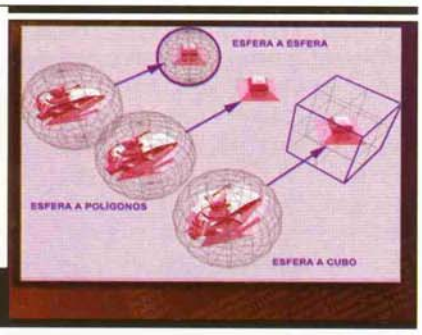
Para terminar, al igual que podíamos obtener la madre de una entidad, con la función *GetChild* podemos obtener el hijo:

```
GetChild (entidad, índice de
la entidad hijo).
```

El "índice de la entidad" corresponde al número de la entidad que queremos. Para saber el número de entidades hijo debemos contarlas con la función *CountChildren(entidad)*.

COLISIONES ENTRE ENTIDADES

La detección de colisiones entre entidades es fundamental para interrelacionarlas y darles una presencia física total en un mundo 3D. En un videojuego, es el proceso que da vida realmente a las evoluciones de la acción. Su tratamiento puede determinar el rendimiento final de un juego, por eso es importante tener bien claro cómo implementarlas. Hay ciertos pasos que hay que seguir para implementar un sistema de colisiones en Blitz3D. Supongamos que queremos detectar la colisión entre dos objetos, por ejemplo, una esfera y un cubo que sirve de pared. En primer lugar, debemos asignarles un número para darlos a conocer al sistema de colisión. Esta operación la realizamos con la función *EntityType* de la siguiente manera:



7

Hay tres métodos de detección de colisiones.

```
Esfera=CreateSphere():
Pared=createcube()
Const ENTIDAD_ESFERA = 1
Const ENTIDAD_PARED = 2
EntityType Esfera, ENTIDAD
_ESFERA ; ← Es igual que
EntityType Esfera,1
EntityType Cubo, ENTIDAD _PARED
```

Utilizamos constantes para facilitar luego la labor de identificar a cada entidad, importante en el caso de que tengamos una enorme lista de objetos a detectar. A continuación asignamos un número a cada objeto.

A continuación, debemos activar la colisión entre los dos tipos de entidades y decirle al Blitz3D qué acción queremos que realice cuando la colisión ocurra. Para ello utilizaremos la función:

```
Collisions entidad fuente,
entidad destino, método de
detección, tipo de acción.
```

NOTA

Varias entidades pueden tener asignada la misma variable de colisión. Por ejemplo, varias esferas diferentes pueden ser del tipo ENTIDAD_ESFERA.

NOTA

Es importante saber que el sistema no realizará ninguna acción después de una colisión hasta que no se actualice el mundo con la función *UpdateWorld()*.

El método que utiliza el sistema de colisiones para la detección de la entidad fuente con otras entidades consiste en rodearla de una esfera imaginaria, la cual servirá para detectar contactos. Es importante, por tanto, decirle al sistema de colisión del Blitz3D el tamaño que tendrá la entidad fuente (es decir, el radio de esa esfera imaginaria) para que pueda ser chequeada. Para proporcionar esta información utilizaremos la función:

```
EntityRadius entidad, radio#
```

B3D utiliza diferentes métodos de detección de colisiones (Fig. 6):

- **ESFERA a ESFERA:** método de detección = 1
- **ESFERA a POLÍGONO:** método de detección = 2
- **ESFERA a CUBO:** método de detección = 3

Cuando nos referimos, por ejemplo, al método "ESFERA a POLÍGONO", queremos decir que la entidad fuente (ESFERA) detectará cada polígono de la entidad destino (POLÍGONO). En este caso, no hace falta decirle al sistema de colisiones el tamaño de la entidad destino porque ya lo sabe.

Sin embargo, si vamos a detectar a la entidad destino con los métodos "ESFERA" y "CUBO", sí es necesario proporcionar el tamaño. Necesitaremos entonces la función:

```
EntityBox entidad, X#, Y#, Z#,
Ancho#, Alto#, Profundidad#.
```

Esta función define el tamaño de un cubo imaginario alrededor de la entidad.

Es preciso comentar que no todos los métodos de colisión funcionan de la misma forma, es decir, unos serán más precisos que otros y más lentos o rápidos. Así el método más preciso y lento es "ESFERA a ESFERA" y el menos preciso y rápido "ESFERA a CUBO".

Además de detectar colisiones, B3D puede generar un determinado tipo de acción cuando esto ocurra. Podemos elegir entre tres: parar, desplazar y caer o desplazar y seguir (Ver ejemplo8_5.bb).

Pongamos un ejemplo:

```
...
EntityRadius Bionave,50
Const ENTIDAD_BIONAVE = 1 :
Const ENTIDAD_EDIFICIO = 2
EntityType Bionave, ENTIDAD
_BIONAVE : EntityType
Edificio, ENTIDAD _EDIFICIO
Collisions Bionave, Edificio, 2, 2
If EntityCollided ( Bionave,
ENTIDAD_EDIFICIO) = True Text
10,10,"Se ha producido colisión"
...
```

En este ejemplo, le decimos al sistema que la entidad "Bionave" detecte a la entidad "Edificio" por el método "ESFERA a POLÍGONOS" y que cuando esto ocurra "Bionave" se desplazará por las paredes. A continuación, con la función:

```
EntityCollided ( Entidad,
Tipo de entidad )
```

podremos saber exactamente si ha habido una colisión específica.

Todas las colisiones ocurridas se van añadiendo a una lista denominada "Lista de información de colisiones". Esta lista es conveniente borrarla cuando no se utilice. Para ello debemos usar la función *ClearCollisions*.

NOTA

El sistema de B3D detecta siempre las colisiones que sufre la entidad fuente con la entidad destino y para ello utiliza siempre el método ESFERA a (método)

En el próximo número...

... aprenderemos a crear y manejar terrenos y nos iniciaremos en el uso de mapas BSP.

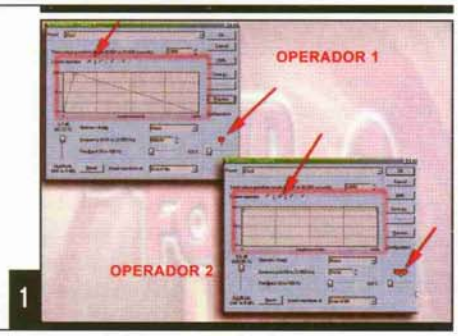
Editores de audio. Sound forge 6.0 (II)

Sound Forge es un programa muy potente con multitud de opciones nuevas y únicas para la creación y manipulación de audio. La semana anterior descubrimos cómo comenzar a trabajar con él y algunos de sus más "íntimos secretos". Cerraremos este tutorial explicando paso a paso cómo crear una fantástica explosión.

A continuación, obtendremos un bucle de sonido para utilizarlo en nuestro secuenciador favorito.

CREAR SONIDOS NUEVOS. SÍNTESIS

Sound Forge 6.0 posee una función bastante interesante que nos permite crear nuevos sonidos a partir de métodos de síntesis, similares a los que se utilizarían en un sintetizador convencional. Accedemos a ella en la opción "Tools \ Synthesis". Disponemos de tres tipos diferentes de síntesis: un generador de tonos (que genera un tono diferente a partir de números y letras introducidos en secuencia en el cuadro "Dial String), un sintetizador FM (Frecuencia Modulada, permite mezclar hasta cuatro formas de ondas diferentes con distintas configuraciones) y por síntesis simple (genera, simplemente, una forma de onda básica con tono y longitud variable). Antes de utilizar cualquiera de estos métodos es necesario crear una plantilla de audio nueva.



Esquema de las gráficas de las ondas utilizadas en la creación de la explosión.

CREACIÓN DE UNA EXPLOSIÓN

Para realizar nuestro sonido desde cero, utilizaremos el generador de síntesis FM. Creamos una plantilla nueva en "File \ New" con formato 44.100 Hz, 16 bits, estéreo. Entramos en el sintetizador FM en "Tools \ Síntesis \ FM". Vamos a utilizar dos operandos con moduladores de ruido o "Noise". Desplazamos el deslizador de configuración hacia la derecha para crear otro operando y colocarlo a continuación del primero (en serie). A continuación, dibujamos las gráficas de cada operando como se muestra en la figura 1.

Para seleccionar cada operando u onda utilizamos las casillas numeradas en "Current operator". Una vez dibujadas las gráficas de cada operando, asignaremos los valores a cada uno de ellos. Para el primero asignamos una duración de salida de 2 segundos en "Total output waveform length", un volumen (Amplitude) del 80 u 82 % y una frecuencia (Frequency) de 500,00. Para el segundo asignamos la misma duración, un volumen del 100% y una frecuencia de 10,00. Pulsando en el botón "Preview" podemos oír el resultado. Elegimos la opción "Start of file" en "Insert waveform at" para colocar el nuevo sonido al comienzo de la plantilla y pulsamos "Ok".

El siguiente paso es expandir el estéreo para lograr más sonoridad. Elegimos la opción "Process \ Pan/Expand". En la gráfica podemos modificar la expansión del sonido a través de los dos canales estéreo. Seleccionamos un volumen (Output gain) de 5.0 dB y el modo "Pan" (preserve stereo separation). La gráfica se muestra en la figura 2.

Esquema de las gráficas utilizadas en la creación de la explosión para el cambio panorámico y de tono.

Para lograr el efecto del impacto debemos modificar el tono del sonido. Para ello, nos valemos del efecto "Effects \ Pitch \ Bend". Aquí podemos dibujar a nuestro gusto la evolución del tono a lo largo del tiempo. Deslizamos la barra de semitonos "Semitones" al máximo, es decir +24, mantenemos deseleccionado la opción "Preserve original duration" y dibujamos la gráfica como se muestra en la

TRUCO

Mientras el previo está sonando podemos modificar la gráfica para oír en tiempo real los cambios.

NOTA

Para mover o crear un punto en las gráficas utilizamos el botón izquierdo del ratón y para borrarlo el derecho.

figura 2. Para conseguir un sonido más grave es imprescindible ecualizarlo. Utilizaremos para ello el ecualizador gráfico situado en "Process \ EQ \ Graphics". Podemos dibujar una gráfica de la ecualización, aunque también es posible utilizar deslizadores por cada banda seleccionando las pestañas "10 Band" o "20 Band" situadas en la parte inferior de la ventana. (Ver Fig. 3).

Antes de terminar siempre es conveniente normalizar el volumen en "Process \ Normalize". Elegimos la opción por defecto "Maximize peak value" y pulsamos "Ok".

Nuestra explosión está casi terminada. Sólo queda darle espectacularidad aplicando un poco de distorsión en "Effects \ Distortion". Situamos "Dry out" en -20 dB y "Wet out" a -3 dB. A continuación dibujamos la gráfica que se muestra en la figura 3.

Sólo nos queda ajustar la duración borrando parte del final y realizando un "Fade out" para realizar una bajada de volumen al final.

Hemos visto que a partir de una simple forma de onda podemos generar cualquier tipo de efecto especial, sólo es cuestión de saber aplicar los efectos y procesos adecuados para buscar lo que se quiere. A continuación, vamos a aprender cómo crear bucles o loops de audio a partir de un sonido.

CREANDO LOOPS

Los loops son sonidos que, al ser reproducidos de forma infinita, parecen no tener principio ni fin. Vamos a partir del efecto de explosión que hemos creado. Hacemos una selección.

Seguidamente, para crear un loop de ahí, seleccionamos la opción "Special \ Insert Sample Loop" o pulsamos la tecla "L". Aparece la ventana "Edit Sample", la cual nos ayudará a configurar nuestro loop a medida. En nuestro caso, es suficiente con pulsar "Ok". La selección que hicimos queda incluida entre dos marcadores, que indican que el loop está creado.

En ocasiones puede ocurrir que el loop creado no comience o acabe realmente bien y en su reproducción continua se note el principio y el fin. Para solucionar este problema disponemos de la función "Crossfade Loop", que permite, por medio de "fades" entre los canales, conseguir un loop perfecto. Es importante saber que para realizar esta operación es necesario que haya espacio antes y después del loop. Elegimos la opción "Tools \ Crossfade Loop". Activamos las casillas "Loop" y "Post-Loop", situamos los deslizadores al máximo (100%) y pulsamos en "Ok". Creamos una plantilla nueva del mismo formato y copiamos el bucle (seleccionado) a ella. Pulsamos CTRL + C, hacemos clic en la nueva plantilla y pegamos con CTRL + V.

Para oír el resultado, activamos la reproducción continua pulsando en el icono correspondiente y a continuación hacemos "Play" (barra espaciadora).

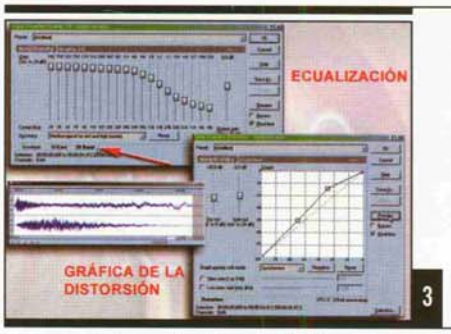
TRUCO

Podemos desplazar un loop creado a través de la línea de tiempo con las teclas " < " y " > ". Si una vez desplazado el loop queremos mantenerlo en la nueva posición, hay que elegir la opción "Update" del menú flotante que aparece al pulsar con el botón derecho del ratón sobre las marcas verdes situadas sobre los límites del loop.

CREANDO UNA LISTA DE SELECCIONES

Sound Forge 6.0 nos permite crear una lista con diferentes selecciones de un mismo sonido o de varios. Podremos guardarla en el disco, reproducirla o utilizarla para crear nuevos sonidos.

Carguemos un sonido, por ejemplo, el de la explosión que creamos anteriormente, y hagamos cualquier selección con el ratón. Seguidamente, elegimos la opción "Special \ Region List \ Insert" para crear una lista nueva. Aparece una ventana de diálogo, donde asignamos un nombre a la selección o región (por defecto va numerado: 01, 02, etc.). Continuamos pulsando en "Ok".



Esquema de la ecualización establecida en la creación de la explosión, así como una gráfica de la distorsión aplicada.

La región queda englobada entre dos líneas de selección y con el nombre que le asignamos. Pulsamos ALT + 1 para mostrar la lista de regiones que hemos creado. En ella, podemos observar cómo contiene un elemento llamado "Selección 1". Si pulsamos en el pequeño icono de su izquierda podremos reproducir sólo esa selección. Además, podemos crear una nueva plantilla automáticamente si desplazamos cualquier elemento de la lista hacia cualquier lugar del fondo.

Hasta aquí este tutorial sobre Sound Forge 6.0. Aunque en líneas generales resulta básico es suficiente para abrir el camino a las funciones más interesantes que esta estupenda herramienta de edición de audio posee.

NOTA

Para poder crear nuevos sonidos a través de la lista de regiones es necesario que todo esté en el mismo formato de audio.

En el próximo número...

... estudiaremos las posibilidades de edición y multipista del Cool Edit Pro 2.0.

Arcades. Shoot´em up

¿Qué jugador no ha tenido entre sus manos alguna vez un matamarcianos? Siguiendo con nuestra pequeña historia de los arcades, toca ahora hablar de los más populares, los *shooters*.

DESCRIPCIÓN

Este tipo de juegos ha estado siempre vinculado a las dos dimensiones y sobre todo a recreativas y consolas de principios de los 90. La estructura general de estos juegos se basa en la temática del método "dispara y olvida". El núcleo principal gira alrededor de una acción dinámica y trepidante, la cual evoluciona linealmente a través de niveles o escenarios que generalmente permanecen constantes. Suelen ser enormemente adictivos, ya que demandan toda la atención del jugador debido al corto tiempo de reacción de que éste dispone durante el juego. Los primeros matamarcianos constaban de un decorado fijo en donde evolucionaban hordas de enemigos a los cuales había que destruir, disparándoles sin tregua. Fueron este tipo de arcades los que encabezaron la historia de los videojuegos, como *Space Invaders, Galaxians o Defender.*

EVOLUCIÓN TÉCNICA

La única evolución que han sufrido ha sido la mejora de gráficos, sonidos y representación en pantalla. Pero, en realidad, siempre han mantenido la misma filosofía: disparar a todo lo que se mueva. Técnicamente, estos juegos se caracterizan por poseer una gran multitud de gráficos simultáneos en pantalla. Habitualmente, en los primeros shooters, el protagonista se encuentra en la parte inferior de la pantalla, cuyos movimientos se limitan a ir de izquierda a derecha. Su misión consiste en disparar para destruir a grupos de enemigos con movimientos prefijados y que a su vez pueden disparar también. La dificultad aumenta a medida que el jugador elimina estos grupos, apareciendo nuevos y más potentes conjuntos de enemigos con diferentes comportamientos. Generalmente, antes de pasar de una etapa a otra, hay que destruir al jefe o enemigo final del nivel.

Con la llegada de las técnicas de scroll simple de pantalla y parallax, el sistema de juego se mantiene, pero la representación de la acción se realiza mediante el desplazamiento horizontal y/o vertical de cada nivel.

Aparecen entonces los decorados interactivos, los cuales se sumaban a las hordas enemigas con elementos de ataque como cañones u otros vehículos. El gé-

NOTA

Los shooters 2D han sido siempre los juegos más abundantes en los salones recreativos. Muchos de esos juegos han sido convertidos a PC o se encuentran disponibles a través de emuladores especiales.

NOTA

Quizás uno de los mejores shooters de scroll horizontal para recreativas sea la serie *R-Type* de la desarrolladora Irem. Para PC, sin lugar a dudas, encontramos la serie de juegos en scroll vertical *Xenon* de The Bitmap Brothers.

1

Imágenes de recreativas de los primeros shooters de la historia.

2

La serie R-Type constituyó el más claro ejemplo de shooter horizontal para recreativas.

3

Xenon 2000 fue el último intento de The BitMap Brothers de shooter con scroll vertical para PC.

LA BIOGRAFÍA...

JOHN CARMACK

Creador del **Shooter 3D**

Junto con Adrian Carmack y John Romero fundó ID Software, compañía creadora del shooter en 3D. Está considerado como el programador de juegos y tecnología 3D en tiempo real más importante e innovador del mundo. Su filosofía es ir siempre de la mano con los nuevos avances tecnológicos en gráficos para ordenador. Gracias a esto se ha convertido en un gurú de los videojuegos con creaciones como la saga *Wolfenstein, Doom* y *Quake*. Es amante de los viejos juegos *arcade* como *Pacman* o *Space Invaders* y aprendió solo a programar. Vendió su primer juego cuando estaba en el instituto, un RPG llamado *Shadowforge*. En 1990 entró a trabajar en SoftDisk, donde programó la serie *Commander Keen*. Su tecnología ha sido muy solicitada y el motor del *Quake II* sirvió para desarrollar juegos como *Soldier of Fortune* o *Half Life*. Carmack es amante de la enseñanza y son conocidas sus andaduras por el mundo dando infinidad de conferencias. Una muestra de esta afición es la posibilidad de adquirir por Internet el código de sus juegos más importantes.

John Carmack.

nero aumenta de variedad, y el desarrollo de al acción transcurre en decorados muy diferentes. Ya no sólo se destruyen naves alienígenas en el espacio exterior, sino también aviones de la segunda guerra mundial, submarinos, soldados o tanques.

Una gran diferencia entre estos arcades con scroll y los de plataformas es que el escenario siempre está en movimiento, obligando al jugador a estar atento en todo momento al desarrollo del juego. A medida que la acción avanza y más enemigos son destruidos, el jugador va obteniendo incentivos, como más potencia de fuego, puntos o vida extra.

Realmente, con la llegada del 3D, la producción de shooters al estilo clásico para PC prácticamente acabó después de la serie *Xenon*. Al igual que ocurría con los juegos de plataformas, sólo se realizan *remakes* de recreativas o consolas. Poco a poco, el 3D comía terreno a los scroll de pantalla en 2D y el género iba adquiriendo otros matices, sobre todo en la jugabilidad, debido a que se añadían cada vez más tonos estratégicos a la acción. Se traslada el género y empiezan a aparecer shoot´em up subjetivos de todas las formas conocidas: combates aéreos, combates de tanques, submarinos o incluso entre soldados. Nace el *First Person Shooter* o *FPS*, encabezado por los legendarios *Doom I* y *II* y *Wolfenstein 3D*.

NOTA

Battlezone (1980) fue el primer shooter realizado en 3D. Ocurrió en las recreativas y simulaba un combate entre tanques con modelos vectoriales.

NOTA

El primer FPS con posibilidad multijugador fue realmente un juego para el Atari ST llamado MIDI Maze, el cual se unía a otro Atari a través del conector MIDI.

La tecnología 3D crece y los FPS se adueñan del mundo lúdico para PC. Tras la aparición y éxito apabullante del juego *Quake* de Id Software (1996), todas las desarrolladoras se suben al carro de este género. Con la llegada de la técnica multijugador con el juego *Doom*, los FPS se transportan a un nivel insospechado de jugabilidad. Ahora, los alienígenas pueden ser creados por el ordenador o ser otros jugadores. Pero a pesar de toda esta tecnología, el argumento de un Shoot´em up mantiene sus cualidades: disparar a todo lo que se mueva.

La perfección de los sistemas multijugador propició la aparición de un nuevo tipo de FPS, los *arena*. En ellos, jugadores conectados a la misma partida luchan entre sí en niveles de pequeña extensión. Destacamos verdaderos juegos de culto como *Unreal Tournament* (Epic Games) o *Quake III arena* (Id Software). El motor gráfico de éste último ha sido muy solicitado para el desarrollo de otros títulos.

Cada vez más, el protagonista adquiere forma humana, dejando atrás naves espaciales o tanques. El género FPS a veces se transforma y entonces la cámara subjetiva se eleva, mostrando al personaje por completo. Aparecen entonces los primeros juegos en tercera persona, que en cierta manera recuerdan a los clásicos shooters de scroll de pantalla. Poco a poco, se irán mezclando factores estratégicos y lógicos en el desarrollo del juego. La incesante acción se vuelve aventura y completar un nivel ya no se reduce a destruir al malo más grande. Sin duda, el juego que marcó un antes y un después en la acción 3D fue *Tomb Raider*, que introdujo el concepto de aventura en los shoot´em up 3D para PC.

En el próximo número...

... hablaremos con más profundidad de este magnífico juego y de los demás arcades 3D que siguieron sus pasos y crearon diferentes subgéneros dentro de los FPS.

Cuestionario Videojuegos 8

Preguntas

1. ¿Qué funciones nos proporciona Blitz3D para mover y rotar una entidad?
2. Describe el procedimiento necesario para que Blitz3D pueda detectar la colisión entre dos entidades.
3. ¿Qué ventajas se obtienen de dividir el código en distintos módulos?
4. Al utilizar un sistema modular de programación debemos definir un número mayor de variables globales, ¿por qué?
5. Antes de poder pintar sobre el modelo 3D en Deep Paint 3D, ¿qué proceso debemos realizar?
6. ¿Cómo podemos pintar con más detalle una pieza del modelo en Deep Paint 3D?
7. ¿Qué proceso debemos realizar en GoldWave si queremos que un sonido suene más rápido?
8. ¿Qué procedimiento debemos realizar para que un sonido termine con una bajada de volumen?
9. ¿Cómo podemos crear un loop perfectamente continuo con cualquier selección en Sound Forge 6.0?
10. ¿Cómo podemos crear en Sound Forge 6.0 una lista de regiones?

Respuestas al cuestionario 7

▷ **1.** GetColor(X,Y)
Rojo=ColoRed(): Verde=ColorGreen(): Azul=ColorBlue()
; Cambiar valor de Rojo, Verde y Azul
Color Rojo, Verde, Azul

▷ **2.** SpriteViewMode Sprite, modo de orientación
Modo 1: El sprite siempre mira a la cámara.
Modo 2: El sprite queda libre de orientación.
Modo 3: Se orienta a la cámara pero no cuando ésta se inclina.
Modo 4: Orientación completa, incluyendo giros e inclinaciones.

▷ **3.**
```
mx#=mx# - Float (MouseYSpeed()) / -7
mx=mx / 1.5
my#=my# + Float (MouseXSpeed()) / -7
my=my / 1.5

X# = EntityPitch ( camara ) + mx
Y# = EntityYaw   ( camara ) + my
```

▷ **4.** Las funciones:
CameraPick (camara, MouseX(), MouseY())
Y PickedX() PickedZ()

▷ **5.** Seleccionando y uniendo sus vértices con la función "Snap".

▷ **6.** En .3DS o .OBJ.

▷ **7.** Para cambiar la frecuencia de muestreo de un sonido.

▷ **8.** Modificando el panorámico por medio de la función "Pan".

▷ **9.** Seleccionando el modo "Create a new window for each take".

▷ **10.** Utilizando la síntesis FM en el menú de herramientas "Tools".

Contenido CD-ROM 8

▶ AUDIO

■ Virtuosa Phoenix Edition 4.01

Excelente reproductor que soporta además múltiples formatos de ficheros de audio.

■ Power Audio Editor 3.0.5

Podremos reproducir, crear, editar y guardar ficheros de audio en diversos formatos y aplicando múltiples efectos.

■ Storm 5.1

Completa herramienta de creación de música, que incluye infinidad de funciones, instrumentos y opciones.

■ MIDINight Express 2.7

Potente reproductor MIDI que emula la reproducción por tabla de ondas.

■ Res Rocket 1.4

Con esta simpática aplicación podrás crear música a través de internet con quien tú quieras.

■ Goldwave 4.6

Nueva oportunidad para hacerte con este excelente programa.

▶ DISEÑO 2D

■ Adesign 1.0

Software para diseñar imágenes en dos dimensiones que además incorpora herramientas para bitmaps y vectores.

■ Ulead PhotoImpact 8.0

Crea logos y otros gráficos con este software que soporta también imágenes en 3D.

■ Image Broadway 4.1

Editor de imágenes que además incorpora un corrector de ojos rojos.

■ Autoimager 2.3

Podrás procesar y convertir imágenes en más de 70 formatos diferentes.

■ CeledyDraw 1.5

Interesante software con el que podrás hacer auténticas virguerías.

■ Pixopedia 24 0.6

Añádele a tus imágenes textos, contornos, formas o movimiento.

▶ DISEÑO 3D

■ Deep Paint 3D 2.0

Demo del popular programa en esta versión que soporta *MercatorUV*.

■ Beyond 3D Extreme 2.01 Beta

Crea mundos virtuales aplicando además texturas y otros efectos.

■ Genesis 3D Editor 1.5

Crea objetos 3D renderizados en tiempo real.

■ Softy 3D 1.0

Aplicación idónea para crear complejos de tipo orgánico de un modo muy sencillo.

▶ PROGRAMACIÓN

■ Quest 3.11

Crea tus propias aventuras conversacionales con la ayuda de esta aplicación.

■ Morfit 3D y Euphoria

Euphoria es un lenguaje de programación sencillo y potente que, con la ayuda de Morfit 3D, nos permitirá programar espectaculares juegos en poco tiempo.

■ Dings Game Basic

Lenguaje de programación diseñado espacialmente para poder programar juegos rápidamente.

■ SCI Studio

Software imprescindible y muy potente para desarrollar juegos en tres dimensiones.

▶ JUEGOS

■ Battlezone

Primer *shooter* en 3D de la historia. Deberás luchar por tener el control de la tecnología alienígena.

■ Doom 1.9

Mata todos los demonios que puedas usando tu arsenal de armas.

■ Unreal Tournament

Magnífico juego con el que podrás competir en red contra todos tus amigos.

■ Wonfenstein 3D

Legendario juego en el que deberás huir de una prisión nazi.

■ Zone of Fighters

Nueva versión de nuestro juego, con muchas más mejoras.

▶ VÍDEO

■ Capture Studio Professional 4.01

Con este útil programa podrás capturar tanto vídeo como imágenes.

■ FadeToBlack 2.0.5

Herramienta que te será de gran ayuda para editar vídeo y ficheros AVI.

■ Full Motion video Pro 4.8

Este editor de vídeo incorpora además múltiples efectos.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.

Visualizando el terreno de juego

Vamos a codificar nuestro juego estudiando los diferentes procedimientos que intervienen en él de forma independiente por medio de la implantación de funciones.

Debido a la extensión del código del juego, sólo explicaremos las partes más importantes, suficiente para comprender todo el funcionamiento. Aun así, en el código fuente se puede encontrar cada una de las partes comentadas con detalle, para que no haya ninguna pérdida en su seguimiento.

Así que empezaremos por un módulo realmente vistoso y espectacular: la creación del entorno de juego y la implementación de todos los aspectos de la cámara.

Las funciones especiales de vídeo y audio las implementaremos en el módulo "funcpantaudio.bb" del juego. Crearemos entonces una función llamada *Crear_entorno()*, que nos servirá para generar el terreno de juego a partir de un mapa de alturas y de sus texturas elegidas en el módulo "menu.bb" (el cual veremos con detalle en otra ocasión), así como de la creación de las luces ambientales, el agua y el cielo.

A medida que definamos variables las iremos incluyendo en el módulo de definiciones "definiciones.bb".

Un buen método de trabajo sería crear las plantillas de todos los módulos y tenerlas cargadas en el editor del Blitz3D. De esta forma, se puede pasar de un módulo a otro con facilidad para añadir datos.

Código 1. Variable global "Tipo_juego"

```
If Tipo_juego=1
 Repeat
  Cls
  Fondo_menu()
  DrawImage logo,20,GraphicsHeight()-100
  DrawImage titulo(5),(GraphicsWidth()/2)-(ImageWidth(titulo(5))/2),20
  Flip
  Locate (GraphicsWidth()/2),150
  Tipo_terreno=Input$(">")
  If Tipo_terreno=0 Then Return
 Until Tipo_terreno>0 And Tipo_terreno<3

 Select Tipo_terreno
  Case 1:
    Terreno$="c:\zone of fighters\gfx\terreno1.bmp"
    textura0=LoadTexture("c:\zone of fighters\gfx\textura1.bmp")
    textura1=LoadTexture("c:\zone of fighters\gfx\textura2.bmp")
  Case 2:
    terreno$="c:\zone of fighters\gfx\terreno2.bmp"
    textura0=LoadTexture("c:\zone of fighters\gfx\textura3.bmp")
    textura1=LoadTexture("c:\zone of fighters\gfx\textura4.bmp")
 End Select
EndIf
```

ELIGIENDO EL TERRENO

Antes de crear el entorno, debemos codificar la parte donde el jugador elige el tipo de terreno. Está elección la implementamos en la función "panel de opciones" del menú de juego.

No entraremos en detalles sobre la estructura del menú porque la veremos en una próxima entrega. Lo fundamental que debemos saber aquí es cómo el ordenador lee un mapa u otro con sus texturas dependiendo de la opción elegida. El jugador dispone de dos tipos de zonas de combates con dos terrenos (mapas de alturas) diferentes por cada zona. Así que lo primero es preguntar qué tipo de juego desea.

ZONA GENERADA ALEATORIAMENTE

En este tipo de juego, el ordenador coloca aleatoriamente, siguiendo unos patrones definidos, todo el decorado, plantas y animales sobre dos terrenos diferentes a elegir. El tipo de juego lo almacenamos en la variable global "Tipo_juego" (Ver tabla adjunta, Código 1).

Básicamente, lo que hacemos es pedir al usuario qué terreno y texturas debemos cargar para el tipo de juego 1 (zona generada aleatoriamente) con la instrucción "Input". Pero antes mantenemos el fondo dibujado llamando a la función *"Fondo_menu()"* e imprimimos la imagen con la frase "Elegir terreno" contenida en título (5) de la matriz de títulos.

Una vez obtenido el dato pasamos a una estructura *"Select .. Case"* en donde, dependiendo del tipo de terreno, asignamos a la variable global *"terreno$"* la situación del fichero del mapa de alturas "terreno1.bmp" o "terreno2.bmp", así como las texturas en "textura0" y "textura1".

ZONA EDITADA

Si el jugador elige la opción de cargar una zona editada previamente con el editor de zonas, debemos obtener el registro que contiene el tipo de terreno que lleva dicha zona. Pero antes, es necesario realizar una serie de operaciones para obtener este dato. Aprovechamos entonces para explicar el proceso necesario para obtener el fichero de datos que contiene toda la zona de combate elegida. En primer lugar disponemos de una matriz llamada *"zona_elegida$"*, la cual contiene todas las capturas de las zonas editadas por el editor. Pero para poder dimensionar con un valor fijo debemos contar cada una de las capturas del disco. Vamos e entonces a crear dicha matriz. Esta operación la realizamos en el módulo "definiciones.bb":

```
dir=ReadDir("c:\zone of
fighters\gfx\datos\")
c=0
Repeat
 f$=NextFile(dir)
 If Left(f$,4)="Img_"  c=c+1
Until f$=""
Dim zona_elegida$(c)
```

Leemos el directorio que contiene las capturas, a continuación recorremos cada fichero e incrementamos el contador "c" cada vez encuentre un archivo que empiece por "Img_". Para terminar, dimensionamos la matriz con el valor del contador.

De vuelta al módulo "menu.bb", haremos la misma operación pero al contrario, es decir, leemos cada fichero con la captura pero para introducir su nombre en la matriz *"zona_elegida$"*:

```
If Tipo_juego=2
  dir=ReadDir("c:\zone of
  fighters\gfx\datos\")
  c=0
  Repeat
   f$=NextFile(dir)
   If Left(f$,4)="Img_"
      c=c+1
      zona_elegida$(c)=f$
   EndIf
  Until f$=""
```

Seguidamente, visualizamos cada captura, previamente reducida a la mitad, en la pantalla. Para ello, entramos en un bucle en donde el jugador, mediante los cursores, pasa de un fichero a otro.

Durante la elección, el índice de la matriz del fichero correspondiente lo almacenamos en la variable local *"cont"*, para luego saber qué fichero cargar. Salimos del bucle cuando se haya pulsado "Return"con nuestro valioso dato almacenado en *"cont"* (Ver tabla Código 2).

Una vez obtenido el nombre del fichero con la zona elegida, debemos cambiar el nombre del fichero contenido en la matriz *"zona_elegida$"*, ya que éste sólo muestra una captura y no los datos que necesitamos. Por lo tanto, aprovechando que diseñamos un nombre de fichero similar para las capturas y los datos, sólo tendremos que cambiar el encabezamiento "Img_" por "Dat_" y la extensión ".bmp" por ".zof".

```
zona_de_combate$=""
zona_de_combate$=zona_elegida$(cont)
zona_de_combate$=Replace
(zona_de_combate$,".bmp",".zof")
zona_de_combate$=Replace
(zona_de_combate$,"Img_","Dat_")
```

A continuación, leemos el fichero con los datos y el primer bloque de registros.

Rechazamos todos los registros (referentes a objetos) menos el último, que contiene el tipo de terreno utilizado para esa zona. Almacenamos este dato en la variable global *"Tipo_terreno"*:

```
fichero=ReadFile("c:\zone of
fighters\gfx\datos\"+zona_de_
combate$)
Null1$=ReadString(fichero)
null2=ReadByte(fichero)
null3=ReadFloat(fichero)
null4=ReadFloat(fichero)
null5=ReadFloat(fichero)
null6=ReadFloat(fichero)
null7=ReadFloat(fichero)
Tipo_terreno=ReadByte(Fichero)
```

Una vez que hayamos obtenido el tipo de terreno utilizamos la misma operación que el tipo de

Código 2. Variable local "cont"

```
cont=1
ze=LoadImage("c:\zone of fighters\gfx\datos\"+zona_elegida$(cont))
ResizeImage ze,GraphicsWidth()/2,GraphicsHeight()/2
Repeat
 Cls
 Fondo_menu()
 If KeyDown(205)
    cont=cont+1
    If cont=c+1 Then cont=1
    ze=LoadImage("c:\zone of
fighters\gfx\datos\"+zona_elegida$(cont))
    ResizeImage ze,GraphicsWidth()/2,GraphicsHeight()/2
 EndIf
 If KeyDown(203)
    cont=cont-1
    If cont=0 Then cont=c
    ze=LoadImage("c:\zone of fighters\gfx\datos\"+zona_elegida$(cont))
    ResizeImage ze,GraphicsWidth()/2,GraphicsHeight()/2
 EndIf
 DrawImage ze,(GraphicsWidth()/2)-(ImageWidth(ze)/2),
 (GraphicsHeight()/2)-150
 DrawImage titulo(7),(GraphicsWidth()/2)-(ImageWidth(titulo(7))/2),20
 Flip
Until KeyDown(28)
FreeImage ze
```

juego anterior para cargar el mapa de alturas y las texturas.

CREANDO EL ENTORNO

Ya tenemos el mapa de alturas y las texturas correspondientes. Así que ya podemos crear el entorno del juego. Volvamos a la función *"Crear_entorno()"* del módulo "funcpantaudio.bb".

CREANDO LAS LUCES

Para poder ver nuestro mundo 3D es necesario iluminarlo. Utilizaremos el comando "AmbientLight" para crear una tonalidad de luz general, la cual afectará por igual a los puntos de cada objeto. Siempre es necesario aplicarla antes de utilizar otras luces. Después crearemos una luz artificial que usaremos como si fuera el Sol.

```
Function Crear_entorno()
variacion_luz#=.05
   luz#=200
   AmbientLight luz#,luz,luz
    light=CreateLight()
   RotateEntity light,45,45,0
   LightColor light,150,150,150
          .....
```

CREANDO EL TERRENO DE JUEGO

Una vez iluminado, podemos crear el terreno a partir del mapa de alturas leyendo el fichero contenido en *"terreno$"*. Según el tipo de terreno, lo escalamos adecuadamente, activamos o no el sombreado según la variable global "shading" y procedemos a darle un número de polígonos según "Detalle_terreno" para crear la forma, además de activar el sistema dinámico de detalle (LOD). Posteriormente, suavizamos los bordes o antialisado según "antialisado ("True/False")".

```
terreno1=LoadTerrain(terreno$)
Select Tipo_terreno
  Case 1: ScaleEntity terreno1,
  30,350,30    ; X=30 x 512
  =15360  Y=350  Z=15360
  Case 2: ScaleEntity terreno1,
  30,380,30
End Select
TerrainShading terreno1,Shading
TerrainDetail terreno1,
Detalle_terreno,True
AntiAlias antialisado
```

El siguiente paso será texturizar el terreno con las texturas contenidas en las variables globales "textura0" y "textura1". La primera textura servirá de piel y dará al terreno los diferentes matices de color según la zona. Como sabemos, la forma que tiene Blitz3D de texturizar un terreno es cubrirlo completamente mediante la repetición continua de la textura (tileado). En esta repetición se utiliza un tamaño de textura por defecto de 1, es decir, el tamaño de una cuadrícula de la malla. Si queremos entonces cubrir por completo el terreno sin la utilización del tileado, debemos escalarla a su tamaño original, es decir, en nuestro caso 512 x 512.

```
PositionTexture textura0,0.,0
ScaleTexture textura0,512,512
EntityTexture terreno1,
textura0,0,0
```

La segunda se utilizará para añadir más realismo a la superficie creando imperfecciones como grietas o rugosidades. Ambas texturas aparecerán mezcladas, predominando la situada debajo. Para esta textura sí podemos jugar con el porcentaje de tileado, variando el escalado. Unas dimensiones de 64 darán buen resultado, pero a medida que la cámara se eleva se notarán más las repeticiones. Así que utilizaremos un valor intermedio de 128 x 128; es decir, se repetirá 16 veces (4 x 4). Luego, la rotamos 45 grados para disimular aún más la repetición.

```
PositionTexture textura1,0.,0
ScaleTexture textura1,128,128
RotateTexture textura1,45
EntityTexture terreno1,
textura1,0,1
```

CREANDO LA URNA, EL CIELO ARTIFICIAL Y EL AGUA

En cuanto al diseño, dijimos que el terrario donde se desarrolla el juego está cubierto por una gran urna. Ésta no es más que un cubo situado de tal forma que engloba todo el terreno.

Conociendo el tamaño del terreno y su posición relativa a la posición 0,0,0, podemos situar perfectamente la urna:

```
urna=CreateCube()
EntityTexture urna,textura_urna
FlipMesh urna
ScaleMesh urna,7500,3000,7500
PositionEntity urna,
7500,1000,7500
```

La sentencia "FlipMesh urna" se utiliza para dar la vuelta a todas las caras del cubo que forman la urna para poder mostrar la textura hacia el interior. Si no lo hiciésemos, no veríamos textura alguna.

CREANDO EL CIELO Y EL AGUA

El cielo es realmente un plano situado 1500 unidades por encima del terreno. Una vez creado lo ocultamos o mostramos dependiendo de si lo tenemos activado desde menú con la variable global "Cielo":

```
mesh_cielo=CreatePlane()
ScaleEntity mesh_cielo,80,20,80
PositionEntity mesh_cielo,0,1500,0
RotateEntity mesh_cielo,180,0,0
ScaleTexture textura2,512,512
EntityTexture mesh_cielo,textura2
If Cielo=True
   ShowEntity mesh_cielo
 Else
   HideEntity mesh_cielo
EndIf
```

En el módulo "Fjuego.bb" tenemos una función para actualizar todo el entorno durante la partida llamada *"Actualiza_Entorno()"*.

En esta función realizamos operaciones como mover el cielo o generar el paso del día a la noche y viceversa. Para mover el cielo realizaremos

una rotación continua de la textura:

```
If Cielo=True
  If angulo_textura#>
  360 Then angulo_textura#=0
     angulo_textura#=
     angulo_textura#+.05
     RotateTexture textura2,
     angulo_textura#
  EndIf
EndIf
```

Y para el paso del día a la noche, rotamos el Sol a la vez que disminuimos la luminosidad tanto de él como del ambiente entre los valores 10 y 200:

```
If Dia_Noche=True
  rotate_luz#=rotate_luz+.05
  RotateEntity light,rotate_luz#,
  rotate_luz,0
  If luz#<10 variacion_luz=
  (variacion_luz)*-1
  If luz#>200 variacion_luz=
  (variacion_luz)*-1
  luz#=luz-variacion_luz#
  AmbientLight luz#,luz,luz
EndIf
```

Volvamos a la función *"Crear_entorno()"* del módulo "funcpantuadio.bb". Para crear el agua debemos saber primero si se ha activado o no la opción del reflejo, ya que dependiendo de una u otra opción, se creará a partir de un plano infinito o de un cubo escalado.

Para poder implementar el reflejo es necesario crear un plano, sin embargo este plano abarcará todo el terreno. La creación de un mirror provoca la bajada del rendimiento del juego en un 50 % ya que prácticamente se renderiza el doble de polígonos.

```
If Reflejo=True
   agua=CreatePlane()
   mirror=CreateMirror()
  Else
   agua=CreateCube()
   Select Tipo_Terreno
    Case 1: ScaleMesh agua,
    4400,4,9000
    Case 2: ScaleMesh agua,
    9000,4,9000
   End Select
 EndIf
```

Una vez creada el agua la posicionamos, le asignamos un color y modificamos el canal alfa para darle un poco de transparencia.

```
PositionEntity agua,8000,25,7000
EntityColor agua,92,64,107
EntityColor agua,40,73,162
EntityAlpha agua,.6
```

(■) CREANDO LA CÁMARA

Para poder observar el entorno es necesario la creación de una cámara, la cual podremos situar en diferentes lugares dependiendo del tipo de vista que queramos: primera persona, tercera persona, cenital o de seguimiento.

Primero, será la variable global *"camara"* la que utilizaremos para manipular la cámara, a continuación es necesario asignarle un campo de visión para delimitar hasta dónde podemos ver. Este concepto es importante porque todo lo que queda detrás del campo de visión no es renderizado. Modificando el "zoom", obtendremos algunos resultados interesantes. Utilizando valores de alejamiento es posible obtener un resultado muy parecido a una lente de ojo de pez. Los valores para asignar un zoom van desde 0 hasta 1, así que podríamos utilizar, para lograr este efecto, 0.54 en la variable global "zoom".

```
cam= CreateCamera()
camara=cam
CameraRange camara,0.1,Rango_vision
CameraZoom camara,zoom
```

Otro aspecto configurable que podemos aplicar a la cámara es la niebla o "fog". Dependiendo del terreno, aplicaremos un color para la niebla u otro, así como para el fondo de la cámara.

Además debemos definir el rango de visualización de la niebla, es decir, lo lejos o cerca del observador. Como ya comentamos en su día, definir el rango de la niebla por delante del rango de visualización de la cámara disimula el corte que sufre el campo de visión (Ver tabla Código 3).

Hemos completado un paso muy importante en la creación de nuestro juego. El siguiente punto a tratar será el estudio del módulo principal y de cómo mostrar en pantalla los indicadores de juego así como el radar de combate.

Código 3. Definición del rango de niebla

```
If Niebla=True
   Select Tipo_terreno
    Case 1:
     CameraFogMode camara,niebla
       CameraFogRange camara,0.0,Rango_vision-100
     CameraFogColor camara,53,87,78
     CameraClsColor camara,53,87,78
    Case 2:
       CameraFogMode camara,niebla
       CameraFogRange camara,0.0,Rango_vision-100
     CameraFogColor camara,130,171,230
     CameraClsColor camara,130,171,230
    End Select
  Else
   CameraRange camara,0.1,20000
EndIf
PositionEntity camara,7624,93,15026
```

En el próximo número...

... seguiremos analizando el código de "Zone of Fighters" empezando por el gestor *de principal*.

Grafismo 3D. Animando nuestro protagonista con Character FX

Vamos a estudiar cómo conseguir dar movimientos a nuestros modelos 3D. Para ello utilizaremos la aplicación Character FX, que nos permitirá conseguir movimientos con gran eficiencia.

Aunque, en principio, en el juego "Zone of Fighters" no implementaremos la bionave con movimiento alguno, su aplicación nos ayudará a comprender el funcionamiento de esta herramienta para poder animar las plantas y animales de la zona de combate en próximas entregas.

CONOCIENDO LA INTERFAZ DE USUARIO

Antes de empezar, vamos a describir las principales partes de la interfaz de usuario de Character FX para tener una idea de su filosofía de funcionamiento.

Para empezar, Character FX trabaja en dos modos diferentes: modo de creación de esqueletos *(objects)* y modo animación *(animation)*. Para seleccionar un modo u otro debemos elegirlo con la paleta de selección de herramientas situada en la ventana principal de la aplicación. En la figura 1 se muestran las partes de esta ventana.

Todo el trabajo que realicemos se muestra en los "Viewports" o vistas. Al ejecutar el programa se activará una vista por defecto, "perspective: 1". Podemos abrir todas las vistas que queramos con la opción "New Viewport" en el menú "Windows" de la ventana principal. Cada una de estas vistas tiene su propio menú de opciones, que veremos a medida que trabajemos con la animación de nuestro modelo. Volviendo a la ventana principal, nos encontramos con una serie de iconos que representan las herramientas disponibles para cada modo de trabajo.

HERRAMIENTAS EN EL MODO "OBJECTS"

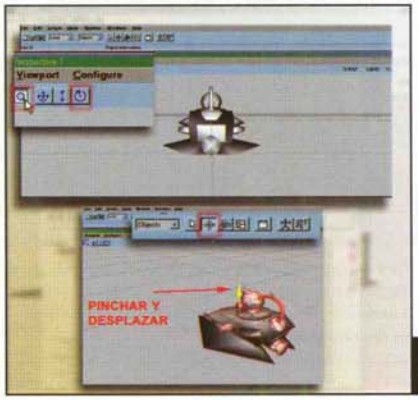
Para trabajar con los modelos y crearles un esqueleto, disponemos de 7 herramientas: herramienta de selección, herramienta para mover, herramienta para rotar, para cambiar de tamaño, herramienta para editar las propiedades de los modelos, herramienta para crear el esqueleto mediante uniones de huesos y herramienta de asignación de vértices para la posterior animación.

Herramienta de selección

Esta opción nos permitirá seleccionar vértices, uniones u objetos completos. Si pulsamos el botón derecho del ratón sobre el icono aparecerá una pequeña ventana de opciones, en la cual podemos elegir qué clase de selección realizaremos. Para seleccionar un objeto debemos hacer clic sobre él en cualquier vista que tengamos abierta. Si queremos sumar más de una selección, seleccionamos mientras mantenemos pulsada la tecla *CTRL* y para eliminar selección la tecla *ALT*. Para borrar cualquier selección haremos *CTRL + Supr*.

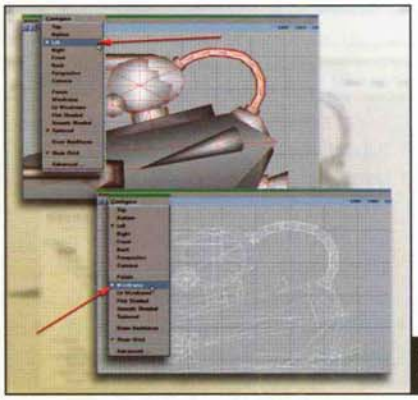
1

Esquema de las partes de la ventana principal de herramientas.



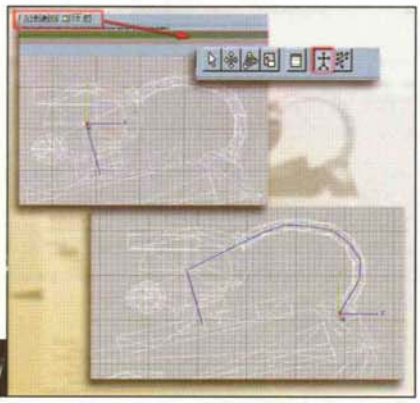
2

Procedimientos para preparar el modelo para la construcción del esqueleto.



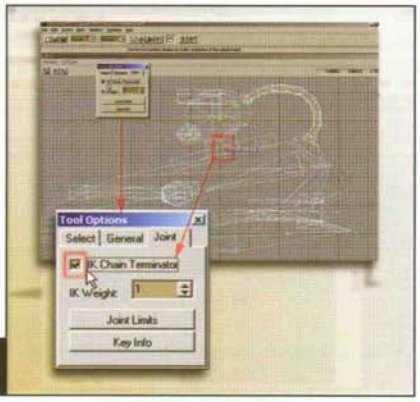
3

En el modo wireframe podemos ver mejor nuestro trabajo de construcción del esqueleto.



4

Con sólo hacer clic, creamos un nuevo punto de unión.



5

Para realizar diferentes esqueletos independientes dentro de un mismo modelo, es necesario asignar, a cada uno de ellos, una unión que sirva de final en la cadena jerárquica.

DEFINICIÓN

▸ CINEMÁTICA INVERSA

Mediante la cinemática inversa se consiguen movimientos realistas de un modelo complejo. Consiste en aplicar más prioridad a la unión del esqueleto situada en la posición más baja de su jerarquía, de tal forma que, al mover este punto, se mueven todos los demás situados a un nivel mayor.

Herramientas de movimiento, rotación y escalado

Para mover una unión o un objeto debemos seleccionarlo primero. Una vez seleccionado se visualizará en su punto de pivotaje los ejes XYZ de coordenadas. Si queremos mover el objeto hacia la derecha o izquierda movemos el ratón haciendo clic sobre la flecha de color roja (Eje X). En la flecha verde (Eje Y) para subir o bajar y en la flecha azul (Eje Z) para acercar o alejar. Para mover libremente, pulsamos sobre la esfera situada en la base de los ejes. Para rotar o escalar el objeto se utilizan los mismos procedimientos. En estas herramientas podemos activar las opciones pulsando con el botón derecho sobre su icono. Observamos que podemos mover, rotar y cambiar el tamaño proporcionando valores exactos.

Editando las propiedades de un objeto

Una vez seleccionado un objeto o una unión, podemos editar sus propiedades con esta opción. Para un objeto, podemos asignarle un nombre y un color. Si elegimos una unión las opciones son más amplias: desde asignar el peso que tendrá dicha unión dentro del esqueleto hasta definirle unos límites de movimientos. Explicaremos más a fondo estas cualidades cuando creemos el esqueleto para la bionave.

Herramienta para crear el esqueleto mediante uniones

Esta herramienta nos permitirá crear un esqueleto completo a partir de la suma de puntos de unión. Con un simple clic en la vista pondremos un punto; si creamos otro en otro sitio, se unirá al anterior y así sucesivamente hasta formar el esqueleto.

Todas las herramientas de selección, movimiento y rotación aplicadas a las uniones se comportan de la misma manera que con los objetos.

Herramienta para asignar y editar vértices

Al activar esta herramienta, el modelo 3D se cubre de vértices en forma de cuadraditos de color verde. Estos vértices se utilizan para asignar una influencia a cada punto de unión; es decir, la unión seleccionada al moverse, afectará a la parte del objeto cuyos vértices hayan sido asignados a dicho punto. Además, a cada asignación le podemos aplicar un peso, el cual definirá la cantidad de influencia que ese punto tendrá en los vértices. Un valor (peso) de 0 indicará una influencia nula.

HERRAMIENTAS EN EL MODO "ANIMATION"

Una vez creado el esqueleto, pasamos al modo de animación para aplicarle movilidad. Encontramos herramientas para mover y rotar los puntos de unión, los cuales afectarán a todo el esqueleto según las asignaciones dadas en el modo "Objects". Además disponemos de una herramienta para mover el esqueleto con el método de cinemática inversa que nos permitirá realizar animaciones más realistas.

EJEMPLO PRÁCTICO. ANIMANDO LA BIONAVE DE COMBATE

Para empezar debemos importar el modelo de nuestra bionave de combate. Vamos a utilizar la que modelamos en su día con Milkshape3D. Si no se tiene a mano, en la carpeta "Extras" del CD que se adjunta se encuentra una copia del modelo 3D en formato .OBJ ("bionave.obj") y su textura ("textura_bionave.png").

Para importar el fichero "bionave.obj" elegimos la opción "File / Import". Una vez cargada la bionave, vemos cómo es mostrada de espaldas en la vista principal. Para ajustar de una forma rápida el modelo a la ventana pulsamos sobre el icono de la lupa una vez. A continuación, movemos la vista pulsando el icono de rotar la cámara y moviendo el ratón haciendo clic sobre cualquier lugar de la cuadrícula de la vista (Fig 2).

También podemos mover la cámara en los ejes X y Z (hacia la derecha e izquierda y hacia delante y atrás) mediante el icono y en el eje Y (arriba y abajo) activando el icono. Una vez situada la vista como se muestra en la figura 2, seleccionamos la herramienta de traslación de la ventana principal y pulsamos sobre el modelo. Al aparecer los ejes, movemos la bionave hacia arriba pulsando sobre la flecha amarilla (eje Y) hasta que quede por encima de la cuadrícula (Fig. 2).

CREANDO EL ESQUELETO

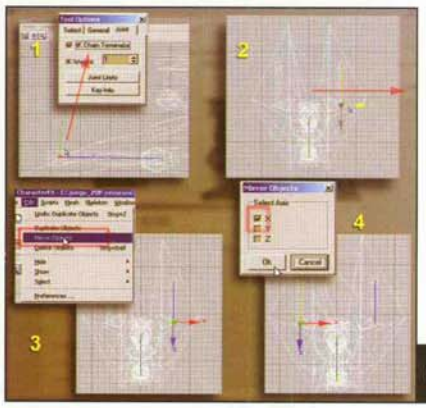
Para crear el esqueleto vamos a situar la vista en proyección lateral izquierda. Elegimos la opción "Left" en el menú "Configure" de la ventana de vista. Acercamos la bionave hacia la cámara mediante el ratón con la herramienta de "Zoom" activada.

Deseleccionamos el modelo activando la herramienta de selección y haciendo clic en cualquier lugar de la vista. Después, conmutamos el modo de representación a "Wireframe" en "Configure" para poder ver mejor las uniones que dibujemos (Fig. 3)

Nuestro propósito es hacer girar la cabeza del protagonista. No debemos olvidar que el casco tiene un tubo flexible que está unido al fuselaje por uno de sus extremos. Así que al mover la cabeza hacia los lados el tubo lo hará también, mientras sigue sujeto al cuerpo de la nave. Necesitaremos entonces crear un esqueleto que recorra el tubo, llegue hacia el centro de la cabeza y siga hasta la base del cuello. Activamos la herramienta de creación de uniones y hacemos clic en el centro del cuello y otro clic en el centro de la cabeza. Observamos cómo los dos puntos se unen. Ya hemos creado un pequeño esqueleto. Vamos a seguir uniendo puntos para recorrer el tubo. Colocaremos un punto nuevo en cada unión de los polígonos del tubo como se muestra en la figura 4.

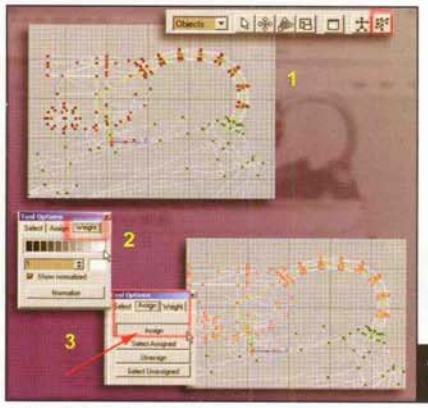
Además de la cabeza, vamos a dar un movimiento de retroceso a los cañones en el momento de disparar. Para ello, debemos crear un pequeño esqueleto formado por dos puntos. Para poder construirlo independientemente del primer esqueleto, debemos convertir el último punto de la jerarquía de éste como punto final de la cadena. Para hacer esto, seleccionamos el punto que servirá de final de cadena y abrimos las opciones de herramientas. Y en la pestaña "Joint" seleccionamos la opción "IK Chain Terminator". Para hacer efectiva la operación debemos elegir la herramienta de selección y pulsar en cualquier lugar vacío de la vista (Fig. 5).

Ya podemos crear un nuevo esqueleto en otro lugar del modelo. En la vista lateral, colocamos un punto en la base del cañón y otro en el extremo y cerramos la jerarquía con el mismo procedimiento anterior. Cambiamos a la vista superior para desplazar el nuevo esqueleto hasta situarlo dentro del cañón. Para crear el esqueleto del otro cañón, basta con copiarlo y desplazarlo. Después



6

Mediante las opciones de copiado y espejo podemos crear otros esqueletos a partir de uno solo.



7

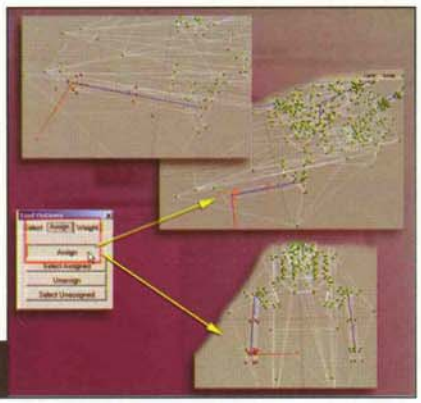
Mediante la asignación de vértices podemos asignar a un punto de unión su influencia en el modelo.

TRUCO

Podemos conmutar las operaciones de copiado y desplazamiento. Si pulsamos el botón copiaremos y si no, desplazaremos.

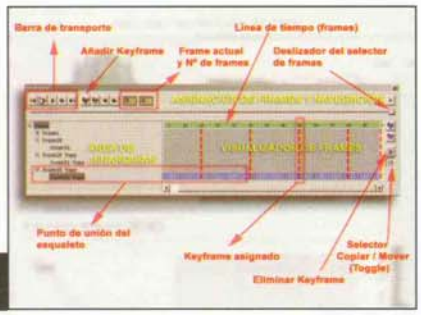
RECORDATORIO

Hay que tener en cuenta que, para animar con Character Fx, es necesario que todas las partes del modelo estén agrupadas formando un solo *mesh*.



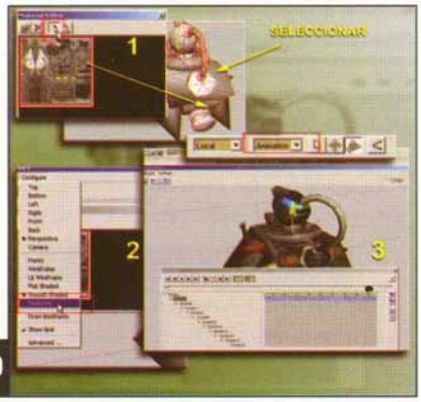
8

Es suficiente con asignar los vértices del cañón a la unión del extremo de éste para realizar un movimiento de contracción.



9

Esquema de las principales partes del Keyframer.



10

Los diferentes procedimientos para aplicar un material al modelo.

NOTA

En la versión comercial de Blitz3D 1.76 o superior es conveniente grabar en formato .B3D pues es muy compatible con todos los métodos de colisión.

tenemos que invertirlo horizontalmente con la opción "Mirror Objects"en el menú "Edit" (Fig. 6).

Una vez creado el esqueleto, debemos asignar a cada unión la influencia que tendrá en el modelo cuando se mueva. Esta influencia se refleja mediante los vértices.

ASIGNANDO VÉRTICES.

Pulsamos en el icono de edición de vértices. Se visualizarán todos los vértices del modelo. Empezamos por el punto principal, el del cuello. Hacemos clic sobre él para seleccionarlo, y a continuación, con mucho cuidado, vamos seleccionando todos los vértices del cuello, cabeza y todos los del tubo. Debemos realizar la operación con la tecla *CTRL* pulsada, para ir sumando selecciones. Cuando elegimos un vértice, éste se volverá de color rojo (Ver Fig. 7).

Una vez seleccionados los vértices, debemos asignarlos al punto elegido. Pulsamos con el botón derecho sobre el icono de selección de vértices. Aparecerá la ventana flotante de opciones.

Pero antes, elegimos un "peso" (influencia) de esos vértices en el modelo de 1 (por defecto) en la pestaña "Weight". Para asignar, pulsamos en el botón "Assign" de la pestaña "Assign". Observamos cómo todos los vértices cambian al color elegido en "Weight", indicando que hemos realizado la selección. Hacemos la misma operación de asignación para ambos cañones como se muestra en la figura 8.

CREANDO LA ANIMACIÓN

Ha llegado el momento de ver si realmente hemos asignado correctamente los vértices realizando el movimiento de la cabeza. Conmutamos al modo "Animation". Al hacerlo aparecerá la ventana "Keyframer" que es donde almacenaremos nuestra animación a lo largo de un determinado tiempo. Su estructura la podemos dividir en tres partes: las herramientas de asignación de frames y navegación, el área de jerarquías y el visualizador de frames. En la figura 9 se muestra una descripción detallada de este editor.

Para probar el movimiento vamos a cambiar de vista y de modo de visualizado. En el menú "Configure" de la ventana de vista elegimos "Smooth Shaded" y "Perspective". Rotamos la vista con el icono de *rotar cámara* para situar el modelo de espaldas. Seleccionaremos el icono de rotación en la paleta de herramientas y pulsamos sobre el punto de unión del cuello. Mientras pulsamos sobre la flecha amarilla (eje Y) movemos el ratón, observamos cómo la cabeza gira llevando consigo al tubo también. Una vez comprobado que todo está correcto, pasaremos a grabar el movimiento en una secuencia de frames. Para observar con más detalles la evolución del modelo durante el movimiento, sería conveniente aplicarle la textura que llevará. Elegimos el modo de visualización "Textured" y seleccionamos el modelo en el modo "Objects". Abrimos el editor de materiales en "Windows / Material Editor". En este editor, pulsamos sobre el icono "Open". Elegimos del directorio "Extras" del CD "Textura_bionave.png" y pulsamos sobre el icono de asignación (Fig. 10).

En el próximo número...

... terminaremos la animación de nuestro personaje y seguiremos creando elementos del juego.

Creando los sonidos del entorno y el menú

Una parte muy importante de nuestro juego son los elementos de audio necesarios para crear ambientación; es decir, todos los efectos de adorno que aumentan la credibilidad de lo que mostramos, como por ejemplo, el sonido de los animales, plantas, el entorno o la voz en off (voz de fondo).

¿Qué sería de un juego sin efectos de audio tales como el rugir de una bestia después de recibir un impacto de nuestro cañón o el clamor del viento en un día nevado? A continuación, vamos a dotar de sonido a los seres vivos que pululan por los terrenos de "Zone of Fighters".

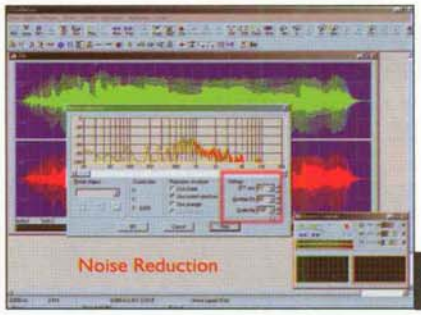
SONIDOS DEL ENTORNO

La idea básica es utilizar nuestros propios medios para conseguir efectos de audio originales. Por ello, explicaremos cómo obtener lo que queremos de una manera "artesanal". Así, para lograr dar vida sonora a los seres que pueblan el juego, utilizaremos nuestra propia voz. Evidentemente, lo que pretendemos es acercarnos lo más posible a nuestra idea, buscando y utilizando los procedimientos adecuados. De modo que nuestra calidad de voz no influirá mucho en el resultado final, pero sí lo hará nuestro trabajo en el editor de audio.

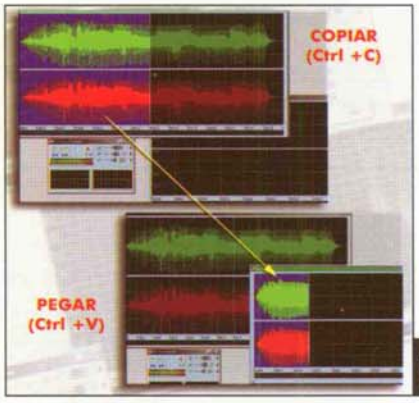
PLANTAS CARNÍVORAS Y GUSANOS

Pretendemos que cuando un *Dreeck* (planta carnívora) o un *Slunk* (gusano subterráneo) reciban el impacto de un disparo reproduzcan un rugido. Vamos a disponer, de base, de la grabación de nuestra propia voz a través de la tarjeta de sonido del ordenador. En primer lugar, y una vez dentro de nuestro editor Goldwave, debemos crear una plantilla de sonido nueva con calidad CD, a 22 Khz. y en estéreo. Recuerda que hay que trabajar de base con buena calidad y luego transformar al formato deseado. A continuación, preparamos el micrófono y asignamos el volumen adecuado para no saturar la grabación. Pulsamos el botón "Record" en "Device Controls" y pronunciamos "AAAH" en tono plano y sin subidas ni bajadas de volumen. Una vez grabado, podemos aplicar un filtro de reducción de ruidos para eliminar la posible basura existente en frecuencias altas (Fig. 1).

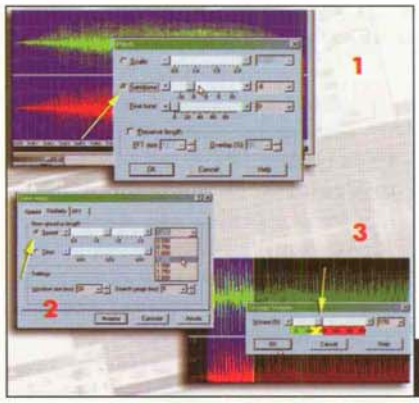
Elegimos la opción "Noise reduction" en "Filter" del menú "Effects" y aplicamos el filtro con los siguientes valores: detalle de análisis del espectro y de la envolvente del ruido (FFT size) en 12, un porcentaje para el cálculo de la envolvente (Overlap) del 80 % y una escala de actuación (Scale) del 100%. Seguimos, aplicando un normalizado, y posteriormente borramos los espacios en blanco del principio y final de la grabación. De todas formas, en el directorio "Extras" del CD se encuentra un ejemplo en formato .WAV llamado "aah.wav". Ya tenemos nuestro sonido base listo para procesar. Vamos a crear



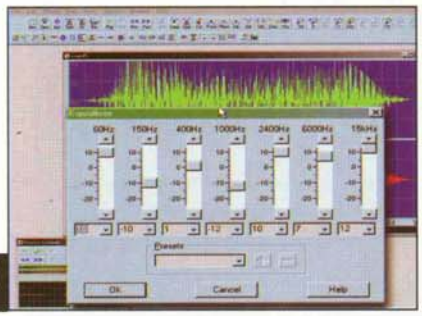
Aplicando un filtro "noise reduction" podemos eliminar ruido en las frecuencias altas del sonido.



GoldWave permite realizar copias de porciones de audio a otra plantilla nueva.

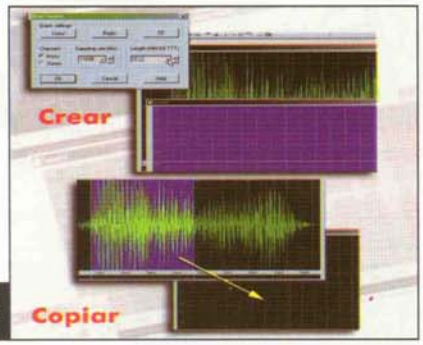


Cambiando el tono (Pitch) a un sonido podemos cambiarlo totalmente.



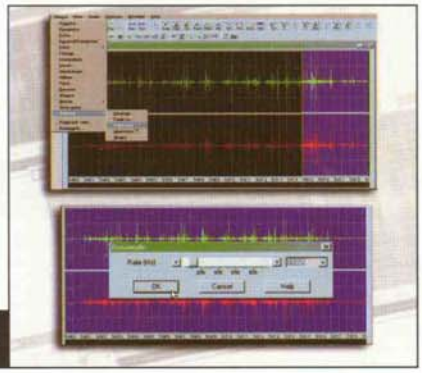
4

La ecualización nos ayudará a obtener un efecto con más sonoridad.



5

Se pueden crear plantillas de audio con distinto formato e intercambiar información de sonido entre ambas.



6

Por medio de la función de resampleo podemos cambiar el frame rate de un sonido.

NOTA

Para grabar la voz en off, crear una plantilla de audio de mucha duración y grabar ahí, de una forma continua, todas las frases y palabras que tendrá el juego para luego ir recortando y procesando individualmente.
Aplicamos un efecto de mecanizado eligiendo la opción "Effects \ Mechanize", designando una calidad del 20%, y normalizando el volumen después.

otra plantilla nueva vacía para ir componiendo ahí nuestro efecto de audio. El primer sonido que crearemos será el rugir lamentoso de un *Dreeck* cuando es alcanzado. Para ello, seleccionamos, más o menos la mitad del audio base. Hacemos *CTRL+C* para copiar, y lo pegamos en la plantilla vacía nueva, situándonos en ella y pulsando *CTRL+V*. Luego, eliminamos todo el sobrante, seleccionándolo y borrando con *CTRL + X*. A partir de ahora trabajaremos en la nueva plantilla (la plantilla base la podemos minimizar) (Fig. 2).

En primer lugar, apliquemos un "Fade in" al principio y un "Fade out" al final. Seguimos, reduciendo 6 semitonos en "Effects \ Pitch" y eligiendo la opción "Time warp" en "Effects" realizaremos una deformación de la duración cambiando la velocidad en la pestaña "Similarity" a un valor de 1.250. Seleccionamos un trozo del sonido para aumentarle drásticamente el volumen con la opción "Effects \ Change Volume" a 170 (Ver Fig. 3).

Volvemos a aplicar un cambio de tono pero guardando la longitud (Preserve length) a -6 semitonos y después normalizamos de nuevo. Para lograr mayor profundidad, aplicamos una ecualización normal con los valores que se muestran en la figura 4.

Para terminar, lo grabamos con la calidad de 8 bits mono a 11025 Hz con el nombre "simpacto_dreeck". Para el rugir del *Slunk* en un impacto utilizaremos el mismo sonido creado para el *Dreeck* pero cambiándole el tono y la velocidad. Creamos una plantilla nueva, esta vez ya con la calidad final, es decir, "Voice" mono a 11025 Hz, y con una longitud de 1. Seleccionamos una parte del sonido el *Dreeck* y lo copiamos a la nueva plantilla (Fig. 5) y a continuación cambiamos de tono y velocidad hasta lograr lo deseado.

Para el *Luny* crearemos un efecto de audio que emule algo viscoso. Un buen método es grabar el roce de una bolsa de plástico grueso o de papel. Preparamos la plantilla nueva con calidad CD y grabamos. Es suficiente con volver a resamplear el sonido a menos calidad (11025 Hz) en la opción "Resample" de "Effects". Añadimos un poco de graves con el ecualizador y grabamos con calidad 8 bits, mono (Fig. 6).

SONIDO DEL MENÚ

En el menú del juego sólo se oye la música de fondo y un pitido le acompaña cada vez que se acepta un grupo de opciones. Esta especie de clic nos servirá de referencia para la opción del menú de cambio de volumen de efectos. Para realizar este pitido, vamos ha utilizar el evaluador de expresiones de Goldwave. Creamos una plantilla nueva con calidad 8 bits, 11025 Hz y mono (Óbice) con una duración de 1. Entramos en el evaluador en "Tools \ Expression evaluator". Utilizaremos una onda sinusoide: escribimos la función *sin(2*pi*f*t)* donde el tiempo (t) vale 0.32 y la frecuencia (f), por ejemplo, 200 (a mayor cantidad más agudo sonará el pitido). Pulsamos en el botón "Start" y, seguidamente, en "Device Controls" (no hace falta cerrar el evaluador de expresiones) pulsamos en "Play" para oír el resultado. Para terminar, nos quedamos sólo con un trozo del audio y salvamos en disco.

En el próximo número...

... empezaremos a estudiar las herramientas necesarias para la realización de la música de nuestro juego.

Creación y manejo de terrenos. Mundos BSP

Antes de empezar con el tratamiento de terrenos queda pendiente el estudio de algunas funciones importantísimas para lograr implementar un sistema de colisiones completo.

FUNCIONES ESPECIALES PARA CONTROLAR COLISIONES

B3D nos proporciona otras funciones muy útiles y necesarias para implementar un buen control de colisiones en nuestros juegos. En ocasiones necesitaremos saber exactamente con qué entidad ha chocado nuestro disparo de entre muchas del mismo tipo. Debemos entonces realizar un procedimiento en el que se ven involucradas algunas funciones muy específicas. Por ejemplo, estamos disparando a un grupo de ovnis, todos del mismo tipo de colisión: ENTIDAD _OVNI. Y queremos saber a qué ovni del grupo se ha alcanzado. Debemos seguir los siguientes pasos:

- **1.** Preguntar si el disparo ha colisionado con un OVNI:

```
If EntityCollided ( Disparo,
ENTIDAD_OVNI ) = True
```

- **2.** Recorrer en un bucle todas las colisiones detectadas en un ciclo de "UpdateWorld" con la función *CountCollisions* (entidad):

```
For n = 1 To CountCollisions
( Disparo )
```

- **3.** Preguntar a cada una de las colisiones detectadas si se trata de una colisión con un tipo ovni. Para ello debemos primero obtener el tipo de colisión de la entidad con la función:

```
GetEntityType ( entidad )
```

Y, a continuación, saber cuál es la otra "entidad" involucrada en la colisión con la función:

```
CollisionEntity (entidad, índice)
```

Donde el parámetro "índice" se refiere al número de la colisión contada con "CountCollisions".

```
Entidad_involucrada=
CollisionEntity (Disparo, n)
If GetEntityType (Entidad_
involucrada) = ENTIDAD_OVNI
...
```

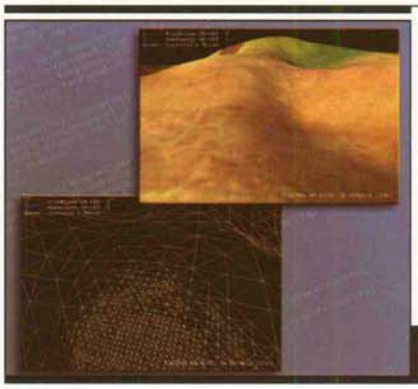
Este procedimiento parece complicado, pero tiene una lógica bastante sencilla. Simplemente, debemos preguntar al Blitz3D las mismas cuestiones que nos hacemos nosotros. Para cada cuestión, obtenemos una respuesta con alguna función. Por ejemplo, una vez que hayamos aislado la "entidad destino" involucrada en la colisión, necesitaremos obtener algunos datos más, como las coordenadas exactas de la colisión o la componente de las normales de dicha colisión, es decir, el lugar y la dirección, por ejemplo, donde recibió el impacto el ovni. Un ejemplo más claro lo encontramos cuando queremos orientar un sprite con la textura de una mancha para simular un impacto en una superficie irregular. Para obtener todos estos datos disponemos de las funciones:

```
CollisionX (entidad, índice)
```

Obtenemos la coordenada X la colisión.

```
CollisionY (entidad, índice)
```

Obtenemos la coordenada Y la colisión.



1

La técnica LOD es bien visible en el modo wireframe (imagen inferior).

```
CollisionZ (entidad, índice)
```

Obtenemos la coordenada Z de la colisión.

```
CollisionNX (entidad, índice )
```

Obtenemos la componente X de la normal de la colisión.

```
CollisionNY (entidad, índice)
```

Obtenemos la componente Y de la normal de la colisión.

```
CollisionNZ (entidad, índice)
```

Obtenemos la componente Z de la normal de la colisión.

Para terminar disponemos de una función que nos permite convertir una entidad en "pikeable", es decir, que puede ser detectada por otra entidad en la distancia. Nos referimos a:

```
EntityPickMode entidad, tipo
de geometría para la detección
```

El "tipo de geometría" funciona de la misma manera que las funciones de colisión:

- **0.** Unpickeable (por defecto)
- **1.** Esfera
- **2.** Polígono
- **3.** Cubo

2

La calidad visual del terreno dependerá también del tamaño de la textura utilizada.

INTRODUCCIÓN A LOS TERRENOS

Blitz3D permite manipular un tipo de objeto 3D formado por multitud de triángulos para la creación de paisajes. Generalmente, estos objetos están formados por cientos de miles de polígonos para poder obtener un resultado adecuado. Sin embargo, dibujar todos estos polígonos conlleva un gran esfuerzo por parte del sistema. Por este motivo, Blitz3D utiliza una técnica denominada LOD, que consiste en cambiar el detalle dinámicamente; de esta manera, se puede mantener una forma aproximada del terreno con menos cantidad de polígonos. (Ver "ejemplo1.bb") (Fig. 1)(/*insertar Blitz3D9_1.jpg) Los terrenos manejados por Blitz3D tienen ciertas características que es necesario tener en cuenta.

En primer lugar, el terreno tiene forma cuadrada y con un tamaño múltiplo de 2, por ejemplo, 2,4 .. 16 .. 512, etc.. Este cuadrado está formado por una malla cuadriculada proporcionalmente y con el mismo número de vértices en cada cuadrícula. Y por último, sólo es posible modificar la altura de un vértice cada vez.

NOTA

En una imagen para un mapa de alturas, las tonalidades negras representan las partes más bajas del terreno y las blancas las más altas.

¿CÓMO CREAR TERRENOS?

Para la creación de terrenos disponemos de dos métodos: utilizando un mapa de alturas o bien por medio de una malla de terreno simple, subiendo o bajando manualmente los vértices.

CARGANDO MAPAS DE ALTURA

Este método es el más eficaz y fácil de utilizar. Consiste en transformar un bitmap en escala de grises en una malla con alturas; es decir, a partir de una malla se generan las alturas dependiendo del tono de gris de la imagen. Es importante saber que, para que se genere correctamente el terreno, la imagen con el mapa de alturas debe ser de proporciones múltiplo de 2, como si de una textura se tratara; es decir, 2x2, 64x64 o 512x512, etc.. (Fig 2.)

A continuación vamos a realizar todos los pasos para crear un terreno a partir de un mapa de alturas.

En el "ejemplo2.bb" se muestra todos estos pasos de una forma práctica (Fig. 3).

En primer lugar debemos cargar la imagen y crear el terreno con la función "LoadTerrain".

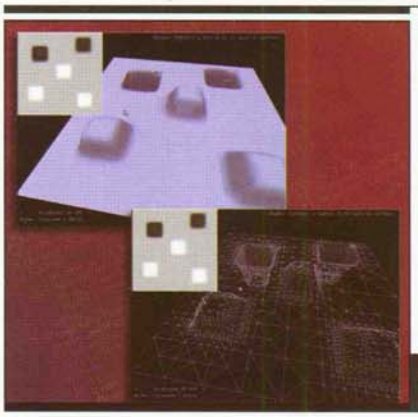
```
Terreno = LoadTerrain
("mapa_altura.bmp"[,madre])
```

Aunque parezca mentira, esto es básicamente todo lo que hay que hacer.

Blitz3D crea el terreno con las dimensiones de la imagen; es decir, si el mapa de alturas tiene 512 x 512 de lado, el terreno se creará con 512 píxels de lado. Sin embargo, la altura será siempre 1.

Para poder cambiar su tamaño, necesitaremos utilizar la función "ScaleEntity".

Por ejemplo: queremos tener un terreno de 15.360 píxels de lado; es decir, en el eje X y en el eje Z, y una altura de 350 con un mapa de 512 x 512. En este caso, escribiremos:



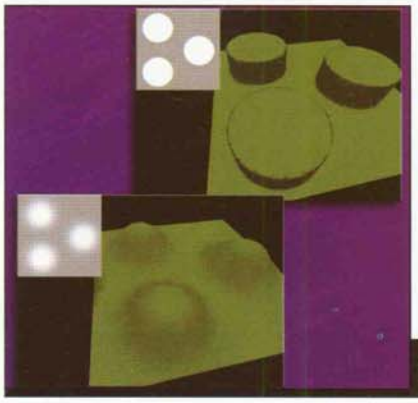
3

Un ejemplo de creación de terrenos utilizando mapas de alturas. Los colores claros representan elevaciones en el terreno y las oscuras hondonadas.

```
Terreno = LoadTerrain
("mapa_altura.bmp")
ScaleEntity Terreno, 30, 350,
30 ➡ 30 x 512 = 15.360
```

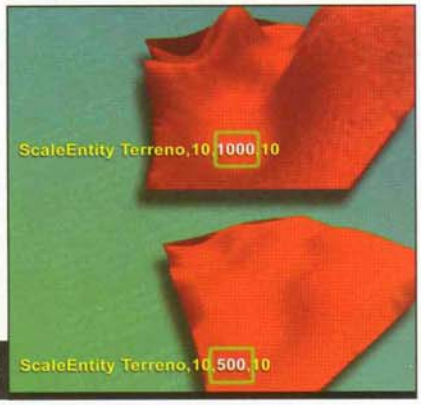
Hay ciertos factores que influyen en la generación de un buen terreno. El principal de todos es cómo dibujemos el mapa de altura. Unos cambios de tonalidad bruscos pueden ocasionar un terreno con filos y cortes, sin embargo, si utilizamos transiciones suaves utilizando desenfoques o *blur* conseguiremos alturas suavizadas (Ver Fig. 4).

Otro factor a tener en cuenta es la altura que le asignemos, escalando la coordenada, ya que a mayores alturas los polígonos



4

Dependiendo de la suavidad en las tonalidades de color en el mapa de alturas lograremos un terreno más o menos redondeado.



5

El escalado del terreno en su eje Y provoca un mayor porcentaje en la representación de las elevaciones.

se estirarán más, dando peor resultado. En la figura 5 se puede observar la diferencia entre dos diferentes alturas dadas a un mismo terreno.

A su vez, hay que tener en cuenta las proporciones en los ejes X y Z a la hora de aportar un valor al eje de altura Y.

CREANDO UN TERRENO A PARTIR DE UNA MALLA

Además de utilizar un mapa de alturas es posible crear un terreno a partir de cero con la función

```
"CreateTerrain": Terreno =
CreateTerrain ( tamaño del
grid, [,madre] )
```

Con esta función crearemos un terreno del tamaño que indiquemos en "tamaño del grid", por ejemplo en: *Terreno = CreateTerrain (512)* Estamos creando un terreno de 512 cuadrículas y de tamaño 512 x 512 x 1 de altura. Hay que tener en cuenta que el tamaño del grid también debe ser siempre múltiplo de dos. Una vez que hemos creado la malla debemos levantar o bajar los vértices para crear las alturas. Debemos recordar que sólo es posible cambiar la altura de un vértice cada vez. Utilizaremos para ello la función *"ModifyTerrain"*:

```
ModifyTerrain Terreno,
Coordenada X del grid,
Coordenada Z del grid, altura
[, tiempo real]
```

En el "ejemplo3.bb" vemos cómo funciona este procedimiento (Fig. 6).

Es muy útil poder saber en cualquier momento qué altura hay en cualquier coordenada del terreno. Para obtener esta información disponemos de la función *"TerrainHeight"*:

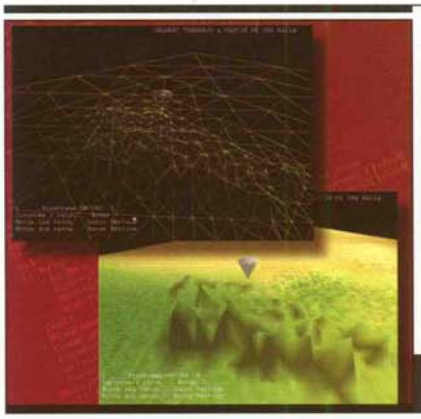
```
TerrainHeight# ( Terreno,
Coordenada X del grid,
Coordenada Z del grid)
(Ver "ejemplo3.bb")
...
Terreno = CreateTerrain (128)
ScaleEntity Terreno,10,50,10
; 1280 x 1280 x 50
Altura# = TerrainHeight#
(Terreno, 500,500) ;
Altura=1280 en X=500 y Z=500
ModifyTerrain Terreno, 500,500,300
Altura# = TerrainHeight#
(Terreno, 500,500) ;
Altura=300 en X=500 y Z=500
...
```

CONTROLANDO EL NIVEL DE DETALLE

Mediante la función "TerrainDetail" podemos controlar el número de polígonos que Blitz3D utilizará para generar el terreno.

```
TerrainDetail Terreno, número
de polígonos (triángulos)
[,transformación de vértices]
```

Sabemos que Blitz3D utiliza el sistema LOD o "nivel de detalle" para reducir los polígonos del terreno situados a gran distancia de la cámara. Esta técnica, a veces, produce la sensación de que el terreno se mueve (sobre todo cuando se utilizan pocos polígonos). Para disimular este inconveniente visual es recomendable activar la opción "transformación de vértices" con "True".



6

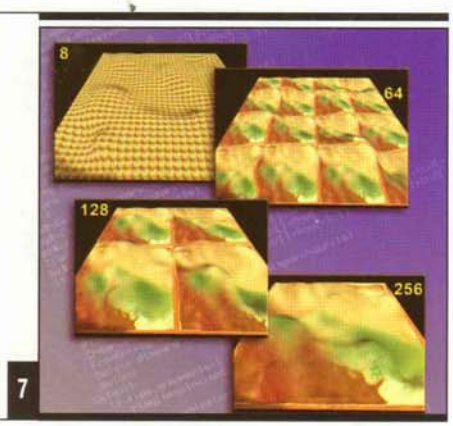
A partir de una malla podemos construir un terreno subiendo o bajando cada vértice.

El número de polígonos está limitado a la potencia del ordenador y sobre todo a la memoria de vídeo disponible (tarjeta gráfica). Evidentemente, la calidad de la representación del terreno, generalmente a partir de mapas de alturas, depende de la cantidad de triángulos que utilicemos. Sin embargo, Blitz3D utiliza una técnica que permite adaptar lo más posible el número de polígonos a la forma original del terreno. Es bueno saber algunos conceptos para averiguar, en cierta medida, la cantidad de polígonos que se necesitarán para obtener un resultado fiel. Por ejemplo, las necesidades poligonales aumentarían si el terreno tuviese demasiadas variaciones bruscas de alturas o si lo escalamos demasiado. También es importante aplicar técnicas para reducir la demanda de polígonos, como por ejemplo: suavizar los bordes en el mapa de altura, reducir el rango de visualización de la cámara o escalar el terreno más allá del límite de visión de la cámara.

También podemos aplicar al terreno sombreado ("shading"), el cual puede cambiar si utilizamos y variamos una luz direccional: *TerrainShading Terreno, True/False.*

NOTA

Se pueden obtener mapas de alturas a través de aplicaciones especializadas como Bryce o bien dibujadas directamente con un programa de dibujo como Paint Shop Pro.



7

Mediante el escalado de la textura controlamos el tileado.

Si usamos este efecto, el rendimiento se reduce, ya que Blitz3D necesita calcular los polígonos afectados por la sombra. Además, visualmente se notaría más el sistema LOD aplicado al terreno.

TEXTURIZANDO TERRENOS

Un terreno se crea con un solo color plano. Así que mediante la función "EntityTexture" podemos asignarle texturas, por ejemplo, para crear el aspecto de la tierra.

El inconveniente radica en que no podemos texturizar partes diferentes del mismo terreno, ya que Blitz3D lo trata como una sola entidad, precisamente por este motivo, la textura lo cubre por completo utilizando la técnica *Mip Mapping*. En el ejemplo:

```
Textura = LoadTexture
("tierra.bmp")
Terreno = CreateTerrain (128)
EntityTexture Terreno,Textura
```

La imagen "tierra.bmp" se aplicará a cada uno de los 128 cuadros que forman la cuadrícula del terreno, formando un mosaico (tileado) de la textura (Fig. 7).

Esto es debido a que, por defecto, se le asigna un tamaño de 1, el cual corresponde a un cuadrado del mallado. Si queremos que sólo una imagen cubra el terreno es preciso escalar la textura al tamaño de éste antes de aplicarla, es decir:

```
ScaleTexture Textura, 128, 128
```

Se pueden obtener resultados realmente brillantes mediante el multitexturizado; es decir, la aplicación de más de una textura al terreno (Ver ejemplo9_4.bb) (Fig. 8).

Es posible aplicar hasta 8 texturas; sin embargo, esta técnica reduciría el rendimiento. Además, esta posibilidad depende de la tarjeta gráfica, ya que no todas soportan tantas mezclas de texturas en el mismo objeto. Generalmente, la posibilidad común es de dos o tres e incluso cuatro en tarjetas de alto nivel como la *GeForce 3* (Fig. 9).

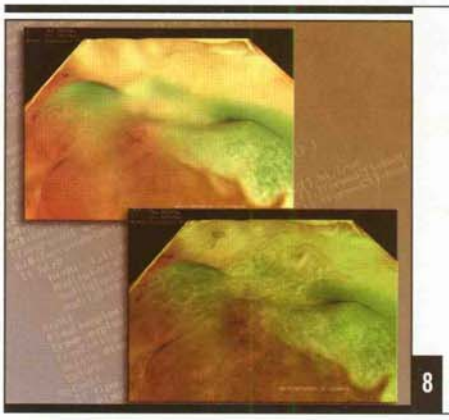
```
Textura1 = LoadTexture ("tierra.bmp")
Textura2 = LoadTexture ("hierba.bmp")
Terreno = CreateTerrain (128)
ScaleTexture Textura1, 128, 128
EntityTexture Terreno,Textura1,0,0 ;
0, 0  frame de la textura y
número de textura
ScaleTexture Textura2, 32, 32
EntityTexture Terreno,Textura2,0,1 ;
0,1 frame 0 y textura número 2
```

Cabe resaltar que es posible emplear diferentes tamaños a cada una de las texturas aplicadas al terreno.

Puede ocurrir que se esté ejecutando una aplicación o un juego que utilice esta técnica de multitexturizado para mezclar tres texturas en un objeto en equipos cuya tarjeta gráfica no soporta más de dos. En este caso Blitz3D emulará la tercera textura, creando una copia del objeto con esa textura. Evidentemente, esto reducirá en gran medida el rendimiento del programa.

OBTENIENDO INFORMACIÓN

En ocasiones podríamos necesitar algunos datos referentes al terreno, como por ejemplo su tamaño o sus coordenadas. Para ello, disponemos de las funciones:



8

Mediante el blending o mezcla de texturas se pueden lograr resultados más realistas.

TerrainSize (Terreno) Retorna el tamaño de la cuadrícula del terreno.

Con las funciones "TerrainX", "TerrainY", "TerrainZ" podemos obtener la posición exacta dentro del terreno.

```
TerrainX# Terreno, X#, Y#, Z#
TerrainY# Terreno, X#, Y#, Z#
TerrainZ# Terreno, X#, Y#, Z#
```

Si, por ejemplo, andamos por el terreno y queremos saber exactamente a qué altura del terreno estamos utilizaríamos "TerrainY".

9

Blitz3D permite mezclar hasta 8 texturas a la vez, sin embargo esta posibilidad depende de la capacidad de la tarjeta de vídeo. Lo normal es mezclar dos o incluso 3 en tarjetas más potentes.

En el próximo número...

... empezaremos con las funciones 3D. Estudiaremos detenidamente el manejo de objetos en 3D.

Editores de audio. Cool Edit Pro 2.0 (I)

Continuamos la serie de tutoriales dedicados a los editores de audio con la segunda versión de Cool Edit Pro de Syntrillium Software.

Este programa tiene la posibilidad de grabar y mezclar hasta 128 pistas de audio en estéreo mediante cualquier tarjeta de sonido compatible con Windows. Pero antes de centrarnos en esta estupenda cualidad, vamos a tener una primera toma de contacto con su interfaz de usuario y aprenderemos a navegar por la ventana de edición y las funciones más importantes.

Al igual que ocurría con Sound Forge, Cool Edit permite el sincronismo de audio con vídeo y además posee más de 40 efectos y procesos de audio en tiempo real, así como herramientas para la restauración de audio en malas condiciones. Otras características son la posibilidad de generar tonos y ruidos, que soporta calidades de hasta 32 bits de resolución y que permite la creación de loops.

PRIMERA TOMA DE CONTACTO

Cool Edit Pro funciona con dos espacios de trabajos importantes: la ventana de edición y la ventana multipista.

Descripción de las principales partes de la ventana de edición.

VENTANA DE EDICIÓN

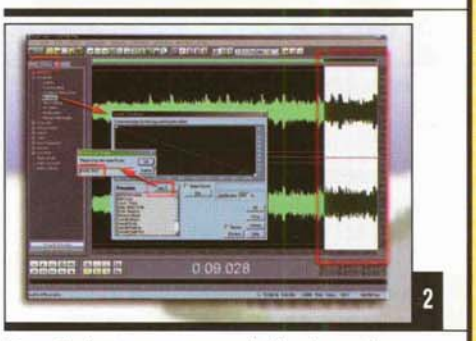
Lo primero que notamos es que la disposición de las ventanas e iconos es más densa que en Sound Forge. Distinguimos una ventana opcional y configurable en la parte izquierda, en la que se encuentra un organizador del trabajo. Podemos ocultarla o visualizarla con *ALT + 9*. En ella podemos tener la lista de todos los sonidos que carguemos, así como los efectos disponibles. El espacio restante lo ocupa la ventana de ondas donde se representa el sonido y donde lo editaremos. En la figura 1 se muestra una descripción de esta ventana.

VENTANA MULTIPISTA

Pulsando en el icono accedemos a la sección multipista del programa. En este espacio de trabajo unimos y mezclamos diferentes sonidos a través de pistas, al igual que lo haríamos con cualquier secuenciador MIDI. Además, podemos aplicar efectos en tiempo real a cada una de las pistas que editemos.

CARGAR Y EDITAR SONIDOS EN LA VENTANA DE EDICIÓN

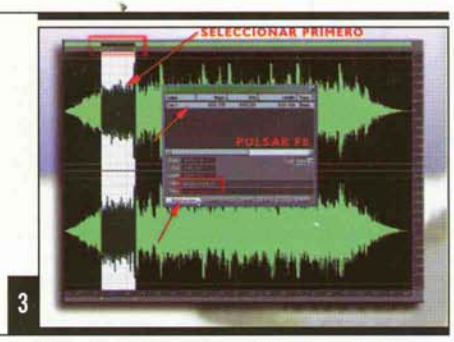
Los procedimientos de carga y edición son muy similares a otros editores de audio, como por ejemplo Sound Forge 6.0. Vamos a realizar algunas operaciones básicas que ya hemos utilizado en ocasiones anteriores para conocer cómo trabaja Cool Edit. Empezamos cargando un sonido contenido en el CD, en el directorio "Extras", llamado "loop1.wav" Esta operación la podemos realizar pulsando en cualquiera de los iconos . Una vez cargado, el siguiente paso será reproducirlo, pulsando en el icono o con la tecla *ESPACIO*. A continuación,



Procedimiento para sumar a la librería un efecto propio. En este caso un FADE.

vamos a aplicar un "Fade in" al principio y un "Fade out" al final. Antes de seguir debemos seleccionar con el ratón la parte final del sonido a la que le aplicaremos el efecto. En Cool Edit no disponemos exactamente de esa función, así que la tendremos que fabricar nosotros. En realidad, un cambio del volumen significa un cambio en la envolvente del sonido, por lo que elegimos el efecto "Envelope" en "Effects \ Amplitude \ Envelope" del menú, o bien seleccionándolo a través del organizador de trabajo, pulsando en la pestaña "effects" y abriendo los menús hasta encontrar "Envelope". Al pulsar dos veces, aparece la ventana "Create Envelope" en la cual se muestra una gráfica y una lista de "Presets". Nuestra intención es sumar a la lista (librería) nuestro Fade Out. Para ello situamos la gráfica como se muestra en la figura 2, que representa una caída del volumen hacia 0.

Una vez dibujada la gráfica pulsamos en el botón "Add" y escribimos el nombre del efecto "Fade Out" y "Ok". Podemos realizar un previo del efecto pulsando en "Preview". El "Fade in" lo haremos de la misma forma pero seleccionando el principio del sonido. Lo siguiente que haremos será, como siempre, normalizar el sonido con



3

Procedimiento para crear listas de selecciones.

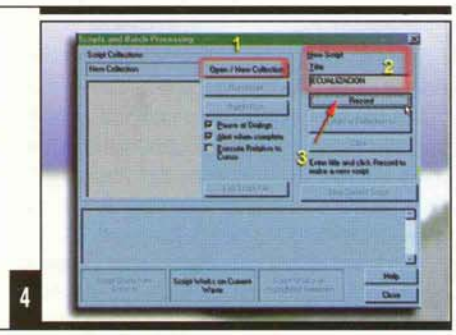
"Normalize" a un 80%. A diferencia de lo que ocurría con Sound Forge, en Cool Edit no podemos tener a la misma vez más de una ventana abierta, aunque sí varios sonidos cargados a la vez con la posibilidad de conmutar entre uno y otro en el menú "Window".

También encontramos aquí la posibilidad de tener una lista de selecciones. Para insertar en la lista es suficiente con crear una selección nueva y pulsar *F8*, pero antes podemos ver la lista pulsando *ALT + 8*. En la figura 3 se muestran los pasos para editar la lista de selecciones.

Si deseamos saber cómo sonará el sonido con otra frecuencia (pero sin llegar a transformarlo) utilizaremos la opción "Adjust Sample Rate" en "Edit"; pero si de verdad queremos convertirlo a otro formato utilizaremos "Convert Sample Rate" en "Edit" o pulsando *F11*.

USANDO SCRIPTS

Una buena utilidad que proporciona Cool Edit Pro es la posibilidad de crear scripts para la grabación de operaciones. Esta opción permitirá almacenar en un fichero de texto los efectos y operaciones exactas que se aplican a un sonido para poder, en un futuro, asignarlas idénticamente a otro sonido diferente. Abrimos la ventana "Scripts and Batch Processing" en "Options". En primer lugar debemos crear un script nuevo y darle un nombre al archivo, por ejemplo "SCRIPT1", y pulsando el botón "Open / New Collection". Una vez creado el nuevo archivo, añadiremos operaciones a la nueva colección de acciones. Vamos a aplicar una ecualización al sonido. Nos situamos en "Title" en "New Script"y escribimos el nombre de la operación, por ejemplo, "ECUALIZACION". Se activará el botón "Record" (Fig. 4).



4

Antes de fabricar cualquier script es necesario crear el fichero que lo contenga.

Una vez pulsado, la ventana se cerrará y el programa quedará a la espera de que se realice la operación de ecualización. Elegimos, por ejemplo, el efecto "Dynamic EQ" en "Filtres". Dibujamos la gráfica como se muestra en la figura 5.

Al pulsar "Ok" el efecto se aplicará. Volvemos a la ventana "Scripts and Batch Processing" y pulsamos el botón "Stop Current Script" para que el programa termine de grabar acciones. Para registrar definitivamente el nuevo script en la colección pulsamos en el botón "<<Add to Collection>>". Podemos editar el nuevo script a través del texto con la configuración generada pulsando en "Edit Script File" (Fig. 6).

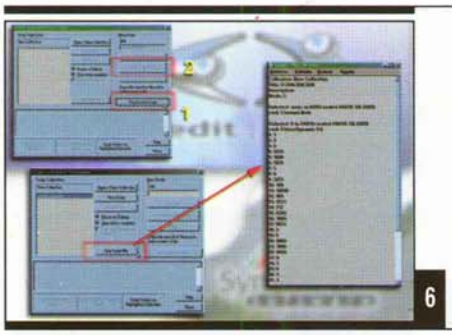
Añadamos otro nuevo script a la colección. En "Title" escribimos "TONO" y pulsamos en "Record". Elegimos el efecto "Pitch Bender" en "Time/Pitch" en el menú "Effects". Modificamos el transcurso del tono en el sonido y aceptamos.

Abrimos la ventana de scripts y realizamos el mismo proceso. Ahora tenemos en disco un archivo llamado "SCRIPT1" que contiene los procesos de ecualización y cambio de tono, los cuales podemos aplicar de la misma forma a otro sonido diferente. Cerramos el sonido que tenemos con *CTRL + W* y cargamos uno nuevo o el mismo. A continuación, entramos en la ventana de scripts y abrimos "SCRIPT1" pulsando el botón "Open / New Collection". Ahora podemos ejecutar cada uno de los scripts de la lista seleccionándolo y pulsando en "Run Script". Se abrirá la ventana del efecto seleccionado, por ejemplo "ecualización", con los mismos parámetros aplicados anteriormente.

5

La ecualización dinámica nos permite dibujar de una forma flexible el ecualizado de nuestro sonido.

Es inevitable observar que el funcionamiento de esta aplicación, sobre todo en los procesos de edición y aplicación de efectos, se asemeja mucho al manejo de Sound Forge 6.0. Sin embargo, el sistema multipista es exclusivo de Cool Edit Pro 2.0 y una herramienta potentísima para la creación de audio. Precisamente explicaremos su funcionamiento, de una forma práctica, realizando la mezcla que hizo posible el audio argumental de "Zone of Fighters".



6

Una vez creado el script podemos editarlo con el bloc de notas de Windows.

En el próximo número...

... aprenderemos las posibilidades del multipista de Cool Edit Pro 2.0

Arcades. Acción/Aventura 3D y subgéneros tácticos

Con la llegada del juego *Lara Croft:Tomb Raider* (Fig. 1), los Arcades 3D adquieren un nuevo concepto: la aventura.

Esta nueva forma de juego permitía al jugador interrelacionarse aún más con la historia, pudiendo, a veces, cambiar su curso dependiendo de su actuación durante el juego. De esta manera, el desarrollo de la partida se hace cada vez menos lineal y aburrido, ya que el jugador necesita resolver, para poder avanzar, ciertos problemas en forma de puzzles, combinaciones de botones o palancas, o por medio de objetos secretos como llaves o claves de acceso. El mundo 3D que el jugador tiene delante se expande en posibilidades, permitiendo una exploración libre del entorno en busca de respuestas.

LA CLAVE DEL ÉXITO, UNA MUJER

Sí, fue el hecho de que la protagonista de Tomb Raider fuera una mujer lo que, en parte, generó el éxito de este juego; además, claro está, del nuevo concepto que introdujo en el género de los Arcades 3D. Cuando, diseñado por Core Design Ltd., fue lanzado en 1996 por Eidos Interactive, sólo se conocían héroes como *Mario* de Nintendo, el "machote" de *Duke Nukem* o el brazo del guerrero de *Doom* o *Quake*. El hecho de transportar la cámara a una tercera persona posibilitó que el jugador adoptara el rol del protagonista del juego. El personaje de la saga *Tomb Raider* tenía nombre propio, *Lara Croft*, e incluso una vida propia.

DEFINICIÓN

▸ VOXEL

Voxel significa crear superficies y modelos 3D por medio de volúmenes creados por píxels. Los voxels pueden tener el tamaño de un punto de pantalla, de una esfera o de un sprite.

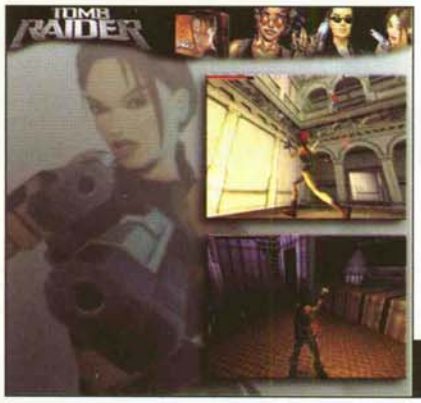
UN MUNDO DE AVENTURAS COMIENZA

El nuevo concepto de aventura aumentaba, aún más, el número de acciones que el jugador podía realizar. Además, se respiraba un cierto aire de Rol en los juegos. Pronto, el nuevo estilo impuesto por *Tomb Raider* inundó las cabezas pensantes de las desarrolladoras que vieron cómo los nuevos Arcades 3D interactivos se hacían dueños del PC. Muchos títulos siguieron estas pautas, como *Outcast* (Infogrames, 1999). Aunque habían pasado algunos años desde el lanzamiento del primer *Tomb Raider*, *OutCast* (Fig. 2) significó un avance técnico importante en los juegos de acción y aventuras 3D, ya que poseía una interactividad absoluta con el entorno y los personajes gracias a la mejor IA conocida hasta el momento.

Además, su motor 3D poseía cualidades gráficas envidiables, como el renderizado de polígonos y voxel, texturas en tiempo real con bump-mapping, sombras y luces dinámicas, animaciones por "Motion Capture" y técnicas cinematográficas para el manejo de las cámaras.

A partir del año 1999 la mayoría de las producciones de acción 3D rondaban en torno a la aventura como *Episodio I: La Amenaza Fantasma* (Lucas Arts,1999), *Prince of Persia 3D* o el original *Messiah* (2000) (Fig. 3).

La proliferación del sistema multijugador como reclamo co-



1

La llegada de Lara Croft: Tomb Raider creó el concepto de aventura en los arcades 3D.

2

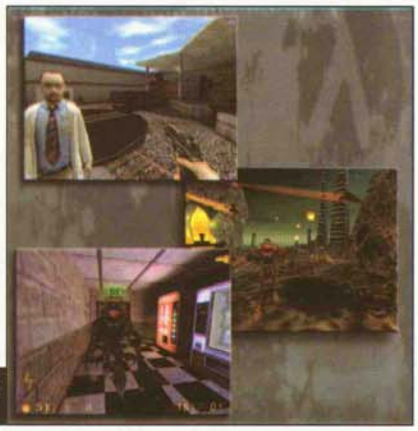
OutCast llevó la aventura a los arcades 3D de una manera única. Permitió al jugador toda la libertad para interactuar con el entorno y los personajes.

NOTA

Aunque la acción 3D de los FPS clásicos cada vez quedaba más a un lado frente a la acción-aventura, aparecieron juegos que, por su calidad gráfica y jugabilidad se han convertidos en clásicos del género como: *Jedi Knight: Dark Forces II* (Lucas Arts, 1997), Unreal (Epic Games, 1998), *Thief: The Dark Project* (Eidos Interactive, 1998), *System Shock II* (Electronic Arts, 1999) o *MDK 2* (Shiny, 2000)

3

Dos ejemplos de auténtica aventura en un juego de acción 3D: Episodio I: La Amenaza Fantasma y Blade: The Edge of Darkness. Este último con posibilidades multijugador para combates DeathMatch (arena).



4

Half Life cambió completamente el concepto de los FPS clásicos. Su concepción poco a poco introdujo el subgénero táctico en los arcades 3D.

NOTA

La inteligencia artificial (IA) de *Outcast* utilizaba el sistema GAIA, el cual modificaba el comportamiento de los personajes a través de agentes. Estos agentes lo formaban la misma acción del juego y eran captados por los personajes, que actuaban según las circunstancias. Se simuló un universo real donde todo estaba relacionado y donde la fauna incluso necesitaba comer, beber o dormir.

mercial importantísimo propició que sistemas de juegos diferentes se unieran en un mismo producto. Es el caso de juegos como *Blade: The Edge of Darkness* (Rebel Act Studio, 2001) o *Rune* (con un sistema acción-aventura para un jugador y un modo arena de combate para multijugador (Fig. 3).

NUEVOS SUBGÉNEROS: FPS Y ARCADES EN TERCERA PERSONA TÁCTICOS

Sin duda, el concepto básico de los FPS se vino abajo con la incursión en el PC del juego *HalfLife* (Sierra, 1998) Este título olvidó los conceptos clásicos en este tipo de juegos, como la activación de palancas o la búsqueda de códigos de acceso para ir resolviendo los niveles. En su lugar, confirió al jugador la tarea de involucrarse en las situaciones como en la vida real, interrelacionándose con los personajes y encauzando los problemas de una manera lógica. Encontramos entonces niveles estudiados al milímetro con puzzles, enemigos inteligentísimos y detalles excepcionales. Y todo desde una perspectiva subjetiva para no olvidar lo que realmente gustaba al jugador del género (Fig. 4).

Este nuevo concepto en los FPS avanzó, y pronto empezaron a salir extensiones de *HalfLife*, como *Counter Strike* (Rookie Studio, 2000) orientado al modo multijugador en el que el jugador forma parte de un equipo de otros jugadores en un fin común, por ejemplo, formar un grupo de policías en busca de terroristas, etc. La calidad gráfica de estos sistemas multijugador no es tan importante, ya que lo que se busca es velocidad, debido al juego masivo a través de internet, y realismo en las acciones y situaciones. Inicialmente, el concepto de táctica de grupo fue introducido en los juegos de acción 3D por la saga *Tom Clancy´s, Rainbow Six* (Red Storm Entertainment, 1998) Pronto, otros títulos adaptaron estos sistemas de juegos como: *Team Factor* (7FX, 2002), la serie *Swat* (Sierra, 2000) o la competencia directa de *Counter Strike, Global Operations* (Barking Dog Studios, 2001), el cual utiliza el motor *LithTech 2.5* de indudable calidad.

Paralelamente, y debido a la gran influencia del género de estrategia en tiempo real que tanto éxito tenía en el mercado, algunos juegos de acción 3D empezaron a mezclar la acción con la estrategia de una forma magistral, como en el caso de la serie *BattleZone* (Activision, 1998) y *BatlleZone II* (1999).

Los combates cuerpo a cuerpo se van sustituyendo por sistemas tácticos organizados por el propio jugador. Ahora, el control se hacía, no de un sólo personaje, sino de todo un grupo de ellos. Este nuevo género táctico también se introdujo con mucho éxito en los arcades 3D con perspectiva en tercera persona y remontándonos a los comienzos debemos hablar sin duda del padre de todos, *Metal Gear Solid* (Konami, 2000). Este título en realidad apareció para la consola PlayStation de Sony. Se trataba de un juego que mantenía el funcionamiento de la escuela impuesta por *Tomb Raider*, es decir, un solo personaje, 3D en tercera persona y sin multijugador. Sin embargo, añadió un nuevo concepto a este tipo de género: la estrategia táctica. Un complejo sistema de seguimiento de la acción por medio de la cámara, un comportamiento totalmente real de los enemigos y unas acciones determinadas para cumplir la misión, llevaron la chispa de la estrategia a los arcades 3D.

No tardaron en aparecer nuevos títulos con este tipo de planteamientos, pero en esta ocasión se basaron en las posibilidades multijugador.

En el próximo número...

... comenzaremos una serie dedicada a la historia de los juegos de estrategias en todas sus vertientes. Empezaremos por la estrategia por turnos.

Cuestionario Videojuegos 9

Preguntas

1. **Disponemos de un mapa de alturas de 256 x 256. ¿Cómo podemos obtener un terreno de 25.600 x 25.600 píxels y 300 de altura?**
2. **Enumera los pasos para texturizar mediante *blending* un terreno.**
3. **Escribe una función que devuelva la raíz cuadrada de un número en Blitz3D.**
4. **Enumera los pasos necesarios para visualizar correctamente un entorno 3D.**
5. **Enumera los pasos para crear una animación en Character FX.**
6. **¿Cómo y dónde deben ir los diferentes movimientos de un personaje en Character FX?**
7. **En GoldWave hemos grabado un sonido a través del micrófono. Al oírlo notamos algún ruido. ¿Cómo podemos solucionarlo?**
8. **¿Qué procedimiento debemos seguir para cambiar la calidad de un sonido en GoldWave?**
9. **En Cool Edit Pro 2, ¿cómo podemos fabricar un cambio de volumen en un sonido?**
10. **Enumera los pasos para crear un script en Cool Edit Pro 2.**

Respuestas al cuestionario 8

▷ **1.** Para mover:

```
MoveEntity Entidad, X#,Y#,Z#
TranslateEntity Entidad, X#,Y#,Z#
PositionEntity Entidad, X#,Y#,Z#
```

Para rotar:

```
RotateEntity Entidad, Pitch(X)#,Yaw(Y)#,Roll(Z)#
TurnEntity Entidad, Pitch(X)#,Yaw(Y)#,Roll(Z)#
```

▷ **2.** Definimos el tipo de entidad con un número:

EntityType Entidad, Tipo_entidad (un número)

Activar el sistema de colisiones y definir la acción en caso de colisión:

Collisions Entidad_Origen, Entidad_Destino, Método_detección, Tipo_acción

▷ **3.**
- Se organiza y estructura todo el código.
- Se facilita el depurado de errores.
- Permite que varios programadores puedan trabajar en el mismo programa.

▷ **4.** Porque, generalmente, son variables compartidas entre varias funciones.

▷ **5.** Antes de poder pintar el modelo en Deep Paint 3D es necesario crear un Mapeado UV para que los vértices de los polígonos del modelo se adapten al dibujo de la textura.

▷ **6.** Podemos pintar con más detalle en Deep Paint 3D, pasando al modo de Proyección.

▷ **7.** Podemos elevar el tono aumentando su escala mediante el efecto "Pitch".

▷ **8.** Aplicando un "Fade Out" en la porción final del sonido previamente seleccionada.

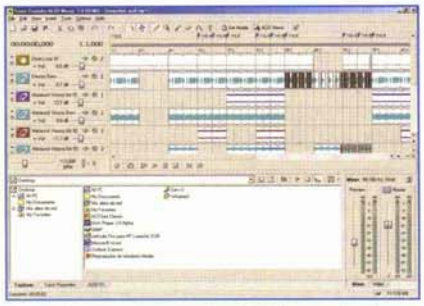
▷ **9.** Mediante la función "Crossfade Loop" es posible crear un loop perfecto en Sound Forge 6.0.

▷ **10.** Seleccionando con el ratón los que queremos añadir a la lista. Seguidamente seleccionamos la opción "Special \ Region List \ Insert". Y finalmente, asignamos un nombre a la selección.

Contenido CD-ROM 9

▶ AUDIO

■ Acid Music 3.0



Completo programa de edición musical con el que podrás realizar auténticas virguerías.

■ MixMeister Pro 4.0

Usa tus archivos MP3 para crear remixes profesionales como un DJ.

■ Screenblast Creation Suite 1.0

Completa suite de creación de contenidos multimedia.

■ Atomix MP3 2.1

Remezcla MP3 en tu PC en tiempo real y usa algoritmos de cálculo de BPM.

■ Quartet X2 Music Studio 2.0

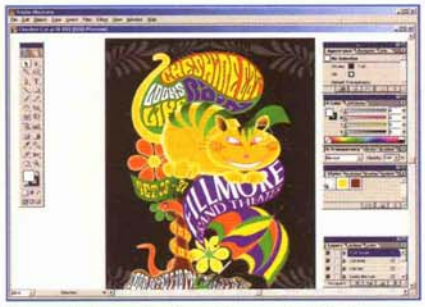
Programa para crear audio en 12 canales (8 de instrumentos y 4 de percusión) fácil y rápidamente.

■ Cool Edit Pro 2.0

Nueva oportunidad para conseguir uno de los editores de audio más utilizado por la gran profesionalidad de los resultados obtenidos.

▶ DISEÑO 2D

■ Adobe Illustrator 10.0



Demo del excelente software de Adobe con el que podrás crear ilustraciones muy profesionales.

■ Liquidpaint 1.0

Crea psicodélicos fondos con esta sencilla herramienta.

■ ColorExact 1.0

Captura los colores exactos de cualquier aplicación.

■ CompuPic Express 6.2

Con este programa podrás crear vistas previas de todas tus imágenes, para poder localizarlas rápidamente.

▶ DISEÑO 3D

■ Effect3D Studio

Crea estupendos gráficos en 3D gracias a esta herramienta.

■ Crystal 3D

Completo programa para crear imágenes e ilustraciones en tres dimensiones.

■ Anything 3D Pano Viewer Pro 1.0

Añádele vistas panorámicas a tus imágenes con esta utilidad.

■ Polytrans 3D

Software profesional con el que podrás crear estupendos modelos en tres dimensiones.

▶ PROGRAMACIÓN

■ Nova Engine 0.3.0

Completo pack de librerías y clases en C y C++ especiales para la creación de juegos.

■ JCad 1.0

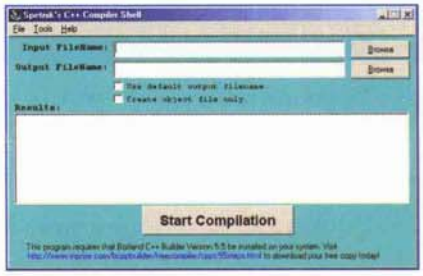
Esta utilidad te ayudará a programar modelos en 3D para tus juegos.

■ FreeCraft Complete 1.17

Creación de juegos de estrategia en tiempo real.

■ Spetnik C++ Compiler Shell 3.0

Compilador de Borland con el que podrás compilar fácilmente C++.



■ Half-Life SDK

Crea tus propios módulos para Half-Life, juego que adjuntamos en este CD-ROM.

▶ JUEGOS

■ Tomb Raider

Acompaña a Lara en su búsqueda y vive aventuras muy emocionantes.

■ Quake

Demo del popular juego que tan buenos momentos nos ha hecho pasar.

■ Half-Life

Intenta sobrevivir en esta experiencia alienígena desde la perspectiva de un soldado.

■ Zone of Fighters

Como cada semana, nuestra nueva versión del juego Zone of Fighters.

▶ VÍDEO

■ NewPlay 4

Podrás reproducir todo tipo de formatos de vídeo usando este programa.

■ Dogma 1.2.2.

Esta potente herramienta es capaz de reproducir múltiples formatos de vídeo y combinarlos con MP3, televisión, etc...

■ FX Movie Splitter 4.5.10

Combina distintos vídeos en uno solo.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.

Módulo principal e indicadores de pantalla

Una vez estudiado cómo "Zone of Fighters" manipula el terreno de juego, le toca el turno a los procedimientos que nos permitirán mostrar todos los indicadores que aparecen durante el mismo. Pero antes vamos a analizar el módulo principal, el cual carga todos los demás módulos y gestiona el bucle principal del juego.

EL MÓDULO PRINCIPAL

En todo juego hay una parte que gobierna el desarrollo de cada partida. Aprovechando la modularidad que nos brinda Blitz3D, vamos a liberar por medio de funciones el desarrollo de este bucle para obtener un código limpio, fácil de modificar y de corregir.

El módulo principal de nuestro juego se llama "zone_of_fighters.bb" y los primeros comandos que debemos escribir en él, y por lo tanto al principio de todo el código, son las definiciones (sobre todo si definimos constantes). Podemos tener todas las definiciones en un módulo aparte, así que debemos incluirlas en el módulo principal con la instrucción "Include". Ya que hemos diseñado un código modular, lo correcto será incluir todos los módulos más o menos por orden al principio del programa.

NOTA

Desde el módulo principal se llama a las demás funciones y es donde debemos compilar nuestro código.

```
Módulo presentación.bb
Función:
Presentación() : pantallas de presentación y argumento
Módulo menu.bb
Función:
Menu() : gestor principal del menú
Panel_opciones() : controla cada una de las opciones
Tabla_Records() : visualiza los records en el menú
Fondo_menu() : visualiza y actualiza el fondo 3D del menú

Módulo Funcarga.bb
Función:
Cargar_texturas_audio(): Carga imágenes, texturas y audio
```

1

Descripción de las funciones de los módulos: "presentacion.bb","menu.bb" y "Funcarga.bb".

```
Include "definiciones.bb"
Include "presentacion.bb"
Include "menu.bb"
Include "funcarga.bb"
Include "funcpantaudio.bb"
Include "decorado.bb"
Include "jugador_principal.bb"
Include "jugadores_CPU.bb"
Include "fjuego.bb"
Include "enemigos.bb"
```

A partir de ese momento se incluirán en el programa principal todas las funciones contenidas en estos módulos.

El siguiente paso es mostrar la presentación de nuestro juego llamando a la función *Presentación()*. Inmediatamente después, llamamos a la función *Definir_Modo_Gráfico()* para establecer el modo gráfico que utilizará el juego a partir de ese momento.

Antes de cargar todas las texturas y sonidos y también antes de crear los modelos y los sprites que servirán de partículas, mostramos un aviso de la carga en la mitad de la pantalla.

Es siempre conveniente tener al usuario informado de todas las operaciones que realiza el programa en tiempos de espera, de esta forma el jugador sabrá si el programa sigue funcionando correctamente.

```
Locate (GraphicsWidth() Shr
1)-200,GraphicsHeight() Shr 1
```

```
Módulo decorado.bb
Función:
colocar_almacen(): coloca almacen en terreno para juego 1
colocar_generador(): coloca generadores en tipo de juego 1
colocar_arbol(): coloca árboles para tipo de juego 1
colocar_puente(): coloca puentes para tipo de juego 1
colocar_roca(): coloca rocas para tipo de juego 1

Módulo jugadores_CPU.bb
Función:
Crear_jugadores_CPU(): crea 5 tipos de ovnis diferentes
actualizar_jugadores_CPU(): controla todo el
                            comportamiento de los ovnis:
                            desplazamiento, ataque y
                            defensa.
```

2

Descripción de las funciones de los módulos: "decorado.bb" y "jugadores_CPU.bb".

```
Print "INICIALIZANDO ZONE OF
FIGTHERS... POR FAVOR ESPERE"
```

"Shr 1" es lo mismo que dividir entre 2. Es una manera más rápida y elegante de realizar una división. Con las funciones *GraphicsWidth()* y *GraphicsHeight()*, lo que hacemos es detectar el tamaño de la pantalla para poder imprimir la frase siempre en medio, independientemente de la resolución de ésta (Fig. 4).

Una cuestión importantísima a tener en cuenta es definir un rendimiento estándar para ordenadores de diferente velocidad, así que debemos crear una variable que guarde el número de fotogramas por segundo (Fps) máximo que pueda correr el juego. De esta forma, nos aseguramos de que en ordenadores superiores el

NOTA

Antes de utilizar cualquier función gráfica es imprescindible tener definido el modo gráfico. Además, cada vez que se defina un modo gráfico nuevo se borrarán de la memoria todas las texturas o modelos que hayamos cargado con anterioridad.

```
Modulo jugador_principal.bb
Función:
control_jugador_principal(): desplaza bionave y ejecuta las
                              acciones del jugador
actualizar_jugador_principal(): actualiza el estado de la
                              bionave en el juego
crear_disparo(): crea un disparo según el tipo elegido
actualizar_disparo(): desplaza el disparo, detecta
                              colisiones y actualiza su estado
actualizar_agujero(): controla visualización marcas agujeros
crear_explosion(): crea un explosión según tipo de disparo
actualizar_explosion(): controla visualizacióne explosiones
escudo_bionave(): controla activación del escudo
camuflaje_bionave(): controla activación del camuflaje
ocultar_bionave(): oculta bionave en vista de 1ª persona
mostrar_bionave(): visualiza bionave en vista 3ª persona
muere_jugador(): visualiza muerte del jugador
```

3

Descripción de las funciones del módulo "jugador_principal.bb".

juego no se ejecutará excesivamente rápido:

```
Timer = CreateTimer ( 60 ) ;
Velocidad máxima en frames
por segundo
```

A continuación, entramos en el bucle principal del juego, el cual contendrá otro bucle para cada partida. El bucle principal será infinito y es sólo a través del menú principal donde daremos la orden para salir del programa. Utilizaremos entonces el bucle "Repeat ...Forever".

Antes de entrar en el menú principal del juego vamos a inicializar el número de luchadores de cada nueva partida y la variable de salida principal "Escape". De vuelta del menú principal, ya tendremos definido, entre otras cosas, el tipo de juego y sabremos si es la primera partida o no (variable "Primer_juego") o si se trata de un juego nuevo (variable "Nuevo_juego"). El contenido de la variable "Primer_juego" es importante para el bucle principal, ya que será utilizada para impedir errores a la hora de llamar a las funciones de borrado antes de llamar a las de creación. La variable "Nuevo_juego" es utilizada para decirle al bucle principal que hemos vuelto del menú con una nueva partida creada o, simplemente, después de haber pulsado "Escape" durante el transcurso de una partida, por lo que impedimos que llame de nuevo a las funciones de creación. Si no lo hiciésemos así, cada vez que pulsemos "escape" durante el juego para ir al menú y "escape" de nuevo para volver al juego, se crearían de nuevo el entorno, los decorados y lo enemigos sobre los ya existentes, doblando así el número de polígonos.

```
Escape=0
numero_luchadores=0
Menu()
If Nuevo_juego=True
  If Primer_juego=False
    Borrar_Entorno()
    Borrar_Decorado()
    Borrar_Enemigos()
    ClearCollisions()
  EndIf
  Crear_Entorno()
  Crear_Decorado()
  Crear_jugadores_CPU(Numero_
  luchadores)
  Mapa_Colisiones()
  Nuevo_juego=False
  Primer_juego=False
EndIf
```

Si es un nuevo juego y es el primero: creamos el entorno, colocamos los decorados, creamos los jugadores enemigos según la cantidad elegida en el menú y almacenada en *Numero_luchadores* y creamos también el mapa de colisiones para todo el juego. A continuación, es importantísimo actualizar las variables "Nuevo_juego" y "Primer_juego" para indicarle al bucle principal la nueva situación.

Si es un nuevo juego y no es el primero (*Primer_juego=False*) debemos entonces borrar el entorno, los decorados, los enemigos y el mapa de colisiones para evitar duplicaciones antes de llegar a las funciones de creación.

Continuamos cargando la música que sonará al final de la partida ("musica2") para evitar tiempos de carga y la que sonará durante la partida, elegidas aleatoriamente de entre dos diferentes ("musica1").

```
musica2=LoadSound("c:\zone of
fighters\audio\musica3.wav")
m=Rnd(4,5)
musica1=LoadSound("c:\zone of
fighters\audio\musica"+Str(m)
+".wav")
c_musica1=PlaySound(musica1)
ChannelVolume c_musica1,
```

4

Con las funciones GraphicsWidth() y GraphicsHeight() es posible determinar el centro de la pantalla en cualquier resolución gráfica.

```
Volumen_Musica
```

Ya estamos preparados para entrar en el bucle que gobierna cada partida del juego. Empezamos estabilizando el rendimiento del juego para ordenadores de diferente velocidad con la instrucción "WaitTimer". Posteriormente detectamos si el jugador pulsa "escape", por lo que saldremos del juego y volveremos al menú. Seguidamente llamamos a las funciones que gobiernan el juego en sí. Actualizamos las colisiones y dibujamos cada escena. Seguidamente mostramos los indicadores de pantalla y el panel de ayuda antes de intercambiar los búferes.

```
Repeat
  WaitTimer(timer)
   If KeyDown(1) Escape=1
   Control_jugador_principal()
   Actualiza_Entorno()
   Actualizar_juego()
   Salvapantalla()
   Pausa()
   UpdateWorld
   RenderWorld
    Indicadores()
    If KeyDown(63)
       DrawImage control,
       (GraphicsWidth()/2)-
       (ImageWidth(control)/2),20
    EndIf
    Flip
  Until Escape=1
```

INDICADORES DE PANTALLA

Una vez visto el funcionamiento del módulo principal del programa, pasaremos a explicar cómo se implementan los indicadores de

5

Según el tipo de disparo seleccionado se visualizará un punto de mira u otro.

pantalla. Vamos a crear una función que se encargue de esta tarea. Se llama *Indicadores()* y estará contenida en el módulo que actualiza todas las acciones del juego: "Fjuego.bb". Básicamente, lo que haremos en esta función es imprimir el estado de la puntuación, munición, vida, armamento, escudo y camuflaje, así como los puntos de mira que aparecen en el modo de primera persona.

Lo primero que haremos es implementar la impresión de estos puntos de mira (Fig. 5).

En función de las irregularidades del terreno la bionave subirá o bajará; por lo tanto, en primera persona la situación del punto de mira variará y no coincidirá con el centro real de la pantalla y como consecuencia no estará centrado con los disparos. Así que debemos variar la altura del punto de mira según esta variación en el terreno, para ello almacenaremos en la variable "y_puntomira" la posición vertical según la coordenada Y del terreno y siempre controlando que no baje de una altura de 100:

```
y_mira=(TerrainY(terreno1,
x,y,z))/4
y_puntomira=(GraphicsHeight()/
2)-y_mira
if y<100 y_puntomira=
y_puntomira+y_mira
```

Seguidamente, dependiendo del tipo de arma que tengamos seleccionada, dibujaremos un punto de mira u otro según "tipo_armamento" (Ver Código 1).

Seguidamente, simplemente definimos las coordenadas para los indicadores, siempre utilizando las funciones para averiguar el tamaño horizontal y vertical de la pantalla, ya que no sabemos en qué resolución se está jugando. De esta manera, podemos dibujar todos los indicadores siempre en el mismo lugar independientemente de la resolución escogida. (Ver Código 2).

Luego, colocamos las imágenes que contienen los gráficos de los indicadores y seguidamente imprimimos sus valores correspondientes. (Ver Código 3).

Todo el trabajo anterior se basa, sencillamente, en ajustar los gráficos y los valores en el lugar de la pantalla en donde se habían prefijado. Según el diseño del juego, debemos imprimir el estado de la vida de los enemigos (plantas, animales y ovnis) en la parte superior de la pantalla cada vez que reciban un impacto. Esta información se

Código 1. Punto de mira según armamento

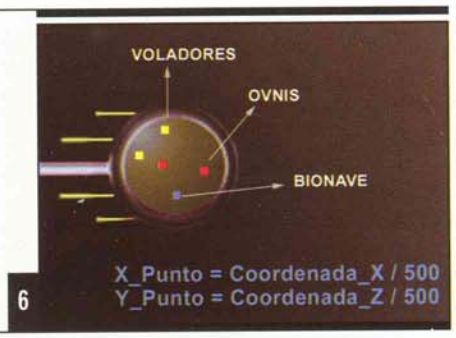
```
If tipo_camara=1 ; Primera persona
  Select tipo_armamento
   Case 1,2  DrawImage punto_mira,(GraphicsWidth()/2)-76,y_puntomira
   Default   DrawImage punto_mira3,(GraphicsWidth()/2)-30,y_puntomira
   End Select
EndIf
```

Código 2. Imágenes con gráficos de los indicadores

```
Xindicador_vidapuntos=GraphicsWidth()-(ImageWidth(indicador_vidapuntos)+20)
Yindicador_vidapuntos=GraphicsHeight()-ImageHeight(indicador_vidapuntos)
Xindicador_armamento=20
Xindicador_camuflaje=GraphicsWidth()-(ImageWidth(indicador_camuflaje))
Yindicador_armamento=GraphicsHeight()-ImageHeight(indicador_vidapuntos)
```

Código 3. Ajuste de los gráficos y los valores

```
DrawImage indicador_vidapuntos,Xindicador_vidapuntos,Yindicador_vidapuntos
DrawImage indicador_armamento,Xindicador_armamento,Yindicador_armamento
DrawImage indicador_escudo,0,Yindicador_armamento-40
DrawImage indicador_camuflaje,Xindicador_camuflaje,Yindicador_vidapuntos-40
Text Xindicador_vidapuntos+115,Yindicador_vidapuntos+22,puntos
Text Xindicador_vidapuntos+12,Yindicador_vidapuntos+20,vida
Text Xindicador_escudo+45,Yindicador_armamento-30,tiempo_escudo
Text Xindicador_camuflaje+40,Yindicador_vidapuntos-30,tiempo_camuflaje
Select tipo_armamento
   Case 1
     If municion<0 municion=0
     Text Xindicador_armamento+35,Yindicador_armamento+22,municion
     DrawImage disparo1B,Xindicador_armamento+154,Yindicador_armamento+16
   Case 2
     If municion2<0 municion2=0
     Text Xindicador_armamento+35,Yindicador_armamento+22,municion2
     DrawImage disparo2B,Xindicador_armamento+154,Yindicador_armamento+15
   Case 3
     If municion3<0 municion3=0
     Text Xindicador_armamento+35,Yindicador_armamento+22,municion3
     DrawImage disparo3B,Xindicador_armamento+151,Yindicador_armamento+20
   Default
     If municion4<0 municion4=0
     Text Xindicador_armamento+35,Yindicador_armamento+22,municion4
     DrawImage disparo4B,Xindicador_armamento+149,Yindicador_armamento+20
End Select
```



6

Esquema del radar del juego y la relación de las coordenadas del terreno (3D) con respecto a las del radar (2D).

visualizará en pantalla durante unos segundos y luego desaparecerá. Para realizar este procedimiento, debemos utilizar una variable que usaremos como contador "tiempo_en_pantalla". Esta variable la inicializamos a 50 cada vez que se produzca un impacto y se irá decrementando en uno cada vez que llamemos a la función de indicadores. De esta manera, mientras no valga 0 seguirá impresa en pantalla la información.

```
If sw_enpantalla=
1 And tiempo_en_pantalla>0
If informacion_vida<
0 informacion_vida=0
  Text (GraphicsWidth()/2),40,
informacion_vida
  tiempo_en_pantalla=
tiempo_en_pantalla-1
EndIf
```

Ya sólo nos falta mostrar el radar y actualizarlo con la función *Radar()* contenida en el módulo "Fjuego.bb".

```
DrawImage radar,0,40
Radar()
```

(■) FABRICANDO UN RADAR PARA EL JUEGO. FUNCIÓN RADAR()

El terreno de combate es amplio, por ello debemos facilitar al jugador la situación de los enemigos mediante la implementación de un sistema de seguimiento.

Se puede construir un radar de muchas formas diferentes. En "Zone of Fighters" utilizamos una serie de puntos de colores que simbolizan nuestra posición y la de los enemigos dentro del terreno de juego. De esta forma, los puntos de color rojo indican la posición de los ovnis, los amarillos, la de los voladores y el azul, la posición de la bionave protagonista.

Para poder representarlos en una pequeña porción de la pantalla (el tamaño del radar), vamos a utilizar las coordenadas de posición de cada uno de los objetos y trasladarlas al espacio del radar. Esto supone dividir el desplazamiento real de cada objeto unas 500 veces (siempre dependiendo del tamaño del radar), es decir: x/500, z/500

Esta división influirá también en la velocidad con que se moverán los puntos por el radar. Irán más rápidos, por ejemplo, con un valor inferior a 500 y más lentos con un valor superior. Hay que tener en cuenta también que si desplazamos los puntos más rápidos, el gráfico del radar debe ser mayor.

7

Visión general de la posición de los indicadores de pantalla en el juego.

Es conveniente saber que la coordenada Z del plano 3D corresponde, en este caso, a la coordenada Y de la pantalla 2D, donde dibujaremos el radar (Fig. 6).

A continuación, debemos desplazar las coordenadas X e Y a la zona superior izquierda de la pantalla, que es donde tenemos situado el gráfico del radar. Así que empezamos dibujando los puntos amarillos que representan a cada uno de los 10 voladores que pueblan el terreno. Para ello, debemos recorrer con un bucle cada volador contenido en la estructura "tipo_voladores" (Ver Código 4).

Seguimos: con los puntos rojos que representan a los ovnis.

Y para finalizar, mostramos la posición del jugador. (Ver Código 5).

```
DrawImage punto_azul,
70-(x/500),50+(z/500)
```

Como hemos comprobado, implementar los indicadores de pantalla ha sido una tarea realmente sencilla (Fig. 7).

Código 4. Bucle que recorre cada volador

```
For volador.tipo_voladores= Each tipo_voladores
      CoordenadasX_volador= EntityX(volador\entidad_volador)
      CoordenadasZ_volador= EntityZ(volador\entidad_volador)
      DrawImage punto_amarillo,70-(Coordenadas_volador
/500),50+(CoordenadaZ_volador/500)
Next
```

Código 5. Representación de los ovnis

```
For jugadorCPU.tipo_jugadorCPU= Each tipo_jugadorCPU
      CoordenadasX_ovni= EntityX(jugadorCPU\entidad)
      CoordenadasZ_ovni= EntityZ(jugadorCPU\entidad)
      DrawImage punto_rojo,70-( CoordenadasX_ovni)/500),50+(
CoordenadasZ_ovni)/500)
Next
```

En el próximo número...

... implementaremos el código necesario para dotar a nuestra bionave de combate de movimiento.

Fabricando los elementos del juego (I)

Antes de empezar a fabricar los elementos del juego debemos terminar de animar la bionave de combate con Character FX.

ANIMANDO LA CABEZA DEL PROTAGONISTA

Nos preparamos adecuadamente acomodando las vistas para poder realizar mejor el trabajo. Si es preciso, abrimos todas las ventanas que necesitemos. En el "Keyframer" pulsamos el botón para ir al primer frame y luego debemos crear el primer "Keyframe" pulsando en el botón para crearlo (Fig. 1).

Desplazamos la barra de navegación hasta colocar la línea de tiempo en el frame 15. Seguidamente, giramos la cabeza hacia la derecha, tal y como hicimos anteriormente, y pulsamos en el botón de "asignar Keyframe". Analizando la operación, hemos creado un punto inicial y otro final para el movimiento de la cabeza. Si reproducimos la animación, observaremos cómo desde el frame 1 hasta el 15 la cabeza gira suavemente hacia la derecha. Continuando con nuestra animación debemos realizar el giro de nuevo hacia el frente. Giramos de nuevo el eje Y y añadimos *Keyframer* (Fig. 2).

También podríamos copiar el primer frame al 30. Para hacer esto, debemos pulsar el botón para conmutar operaciones de copiado y desplazamiento para activar el modo de copiado (pulsado son operaciones de copiado y no pulsado de desplazamiento) Luego, seleccionamos, englobando con el ratón, toda la barra roja del frame 1. Ésta queda rodeada por un rectángulo amarillo. Posteriormente, nos situamos sobre este rectángulo y lo desplazamos hacia el frame 30 con el ratón. Observamos cómo el frame 1 se ha copiado en el 30.

A continuación, debemos grabar el giro hacia la izquierda, añadiendo más frames libres a la línea de tiempo y realizando las mismas operaciones anteriores (Fig. 3).

NOTA

Si nos quedamos sin frames libres en la línea de tiempo, podemos crear más, escribiendo el número de frames en la casilla de"frame final" y pulsando "Enter".

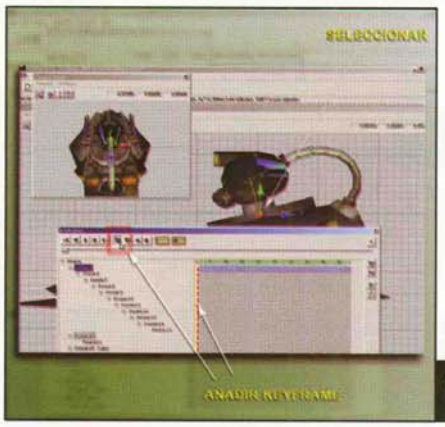
NOTA

Es importante saber que las diferentes animaciones deben ir unidas secuencialmente. De esta forma es posible extraerlas desde programación indicando los frames de comienzo y final de cada una.

ANIMANDO LOS CAÑONES DE LA BIONAVE

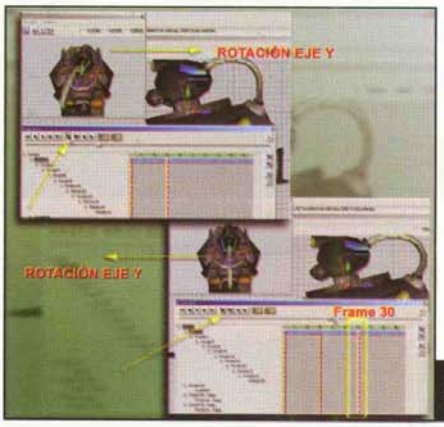
El último paso de nuestra animación es crear un movimiento de retroceso para ambos cañones a la vez. Añadimos 10 frames más a la línea de tiempo, suficientes para realizar el retroceso y nos situamos en el frame 70.

Seleccionamos el punto del extremo del cañón izquierdo y lo desplazamos hacia atrás un poco. A continuación, grabamos el "Keyframe". Nos situamos en el frame 75 y volvemos a desplazar el punto del extremo a su po-



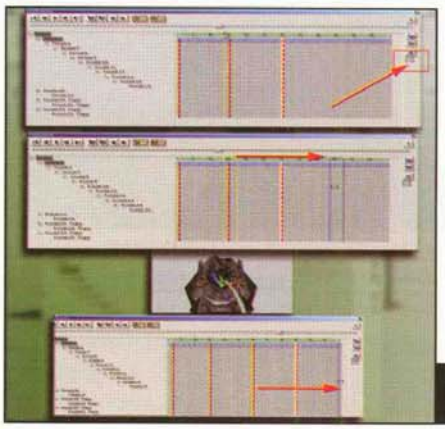
1

Procedimientos para empezar una animación en el modo "Animation".



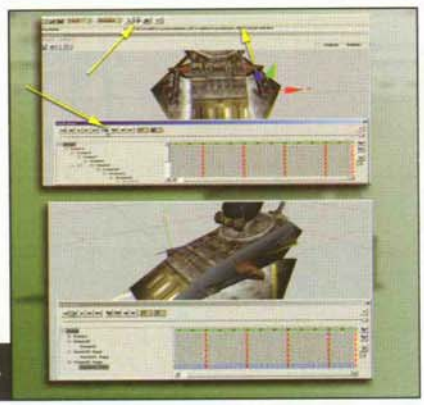
2

Añadiendo keyframes podemos almacenar los diferentes cambios en la animación.



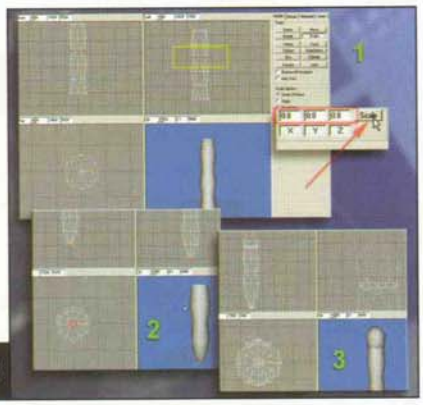
3

Es posible copiar y mover keyframes en la línea de tiempo de la animación.



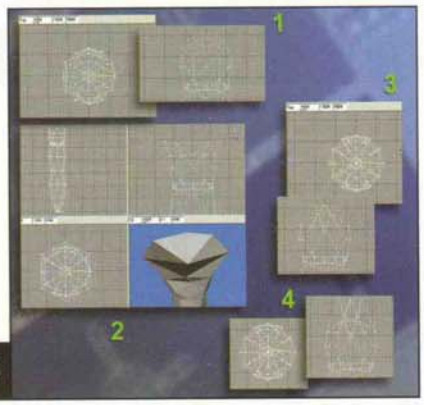
4

En un mismo keyframe podemos añadir diferentes movimientos para los distintos esqueletos de un mismo modelo.



5

Procedimientos para el modelado del tronco del Slunk. Los anillos nos ayudarán a dar forma al tronco.



6

A partir de una geoesfera y desplazando vértices podemos obtener la forma de la boca.

sición original. De nuevo pulsamos el botón de grabación de "Keyframe" (Fig. 4).

Es posible añadir algunos movimientos de detalle más a la animación, por ejemplo, un pequeño giro hacia delante de la cabeza en el momento del disparo. Para ello, nos situamos en el frame 70 y giramos un poco la cabeza. Para añadir este nuevo movimiento sólo tenemos que pulsar de nuevo en el botón de grabación de "Keyframe". Desde el frame 65 al 70 la cabeza volverá automáticamente a su posición inicial, así que no es necesario volver a girarla.

GUARDANDO LA ANIMACIÓN

Una vez completada la animación de nuestro modelo, nos queda exportarla a disco para su utilización en Blitz3D. Esta operación la podemos realizar en varios formatos, desde .3DS y .X, hasta .MD2 y el formato estándar de Blitz3D .B3D.

Si disponemos de la versión comercial de Blitz3D 1.76 o superior, es conveniente grabar en formato .B3D por su manejabilidad posterior a la hora de implementar las animaciones mediante programación y por la total compatibilidad con todos lo métodos de colisión.

Si por el contrario trabajamos con versiones inferiores (versión demo), las cuales no soportan este nuevo formato, nos tendremos que conformar con utilizar el formato .MD2, muy manejable también, pero no compatible con todos los métodos de colisión.

Para exportar simplemente seleccionamos la opción *Exporter* del menú *File* y elegimos el formato adecuado, aceptando todos los parámetros por defecto.

FABRICANDO LOS ELEMENTOS DEL JUEGO

Continuamos nuestro desarrollo gráfico empezando la realización de los demás elementos del juego, desde los animales y plantas hasta todo el decorado. Para esta ocasión no utilizaremos la herramienta MercatorUV de Deep Paint 3D para realizar el mapeado UV de los modelos. En su lugar trabajaremos con la aplicación LithUnwarp, ya que precisaremos de un mapeado simple. Además, esta herramienta nos ayudará a optimizar de polígonos nuestros modelos. Una vez optimizado y mapeado en LithUnwrap, salvaremos el modelo 3D en formato .OBJ para luego pasar al Deep Paint 3D y texturizarlo. Pero antes debemos empezar por modelar cada uno de los elementos. En esta entrega realizaremos el gusano gigante *Slunk* y la planta *Luny*.

MODELADO DE UN GUSANO GIGANTE

CABEZA Y CUERPO

Ejecutamos MilkShape 3D y preparamos las vistas. La forma del cuerpo de los *Slunks* es básicamente cilíndrica, así que partiremos de un cilindro de 6 anillos (stack) y 12 divisiones (Slices). Cada una de las partes nos servirá para poder dar un poco de forma al cuerpo, así que, empezando por abajo, seleccionamos la fila segunda de vértices (con la opción "Select Backfaces" deseleccionada) y escalamos todos sus ejes con un valor de 0.8. Haremos lo mismo con las filas cuarta y sexta. Para crear la cola y así cerrar el cilindro, seleccionamos la primera fila de abajo y escalamos manualmente en la vista "top" hasta unir prácticamente todos los vértices entre sí. El siguiente paso es la cabeza, la cual haremos a partir de una geoesfera de densidad 1, es la que menos polígonos tiene y resulta ideal para modificar los vértices. Para finalizar, la ajustamos al tronco con la herramienta de escalado (F4) (Fig. 5).

Para dar la forma a la boca, seleccionamos las 3 filas de vértices superiores de la cabeza

y con la herramienta de escalado la abrimos. Seguidamente, seleccionamos el único vértice superior y lo bajamos como se muestra en la figura 6.

Siguiendo con la forma de la boca, seleccionamos los vértices que forman pico y los subimos para alargar la mandíbula. Posteriormente los seleccionamos y escalamos hacia dentro.

LAS PATAS

Vamos a añadir cuatro patas que situaremos en la base de la cabeza. Cada una de ellas está formada por dos cilindros. Uno de ellos de 1 "Stack" y 6 "Slices" y otro con 3 "Stack" y 6 "Slices". Al segundo cilindro le desplazamos los vértices de unión de cada división ("Stack") para obtener una forma más o menos curva. Para modelar el extremo de la pata, seleccionamos los tres vértices superiores (siempre con "Select Backfaces" deseleccionado) y los unimos con "Ctrl" + "N". Queremos obtener una forma más puntiaguda, así que desplazamos estos vértices: los inferiores para abajo, los del centro hacia fuera y los superiores hacia arriba (Fig. 7).

Terminamos, uniendo las dos partes de la pata por medio de sus vértices con "Ctrl" + "N" como se muestra en la figura 8 y colocándola en su lugar por debajo de las mandíbulas. Para realizar las siguientes patas, sólo es necesario hacer copias y aplicar procedimientos de espejo ("Mirror").

Seleccionamos las dos partes de la pata con la tecla "Shift" pulsada, a continuación la duplicamos con "Ctrl" + "D" y realizamos un "Mirror Back <-> Front". Para las otras dos patas, duplicamos, rotamos su eje Y en 90 grados y luego colocamos, duplicamos de nuevo y hacemos un "Mirror Left <-> Right"

Antes de guardar nuestro modelo debemos agrupar todas las partes para formar un solo objeto 3D. Para ello, las seleccionamos todas y en la pestaña *Groups* pulsamos el botón "Regroup".

Terminamos exportando en el formato de Milkshape 3D .MS3D en el directorio "C:\juego_ZOF\Slunk\Modelado" con el nombre "Slunk.ms3d".

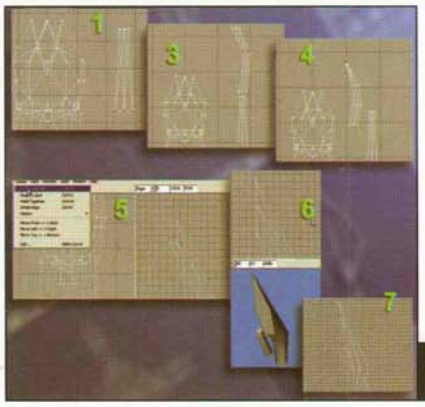
CREANDO EL MAPEADO UV CON LITHUNWRAP

Antes de texturizar el modelo, debemos generar un mapeado UV para que Deep Paint 3D pueda pintarlo correctamente. En vez del MercatorUV utilizaremos la herramienta LithUnWrap.

Este programa nos permite optimizar nuestros modelos y generar un mapeado UV del mismo. Una vez dentro de la aplicación cargamos el modelo "Slunk.ms3D" anterior en "File / Model / Open". Podemos observar en la pantalla un entramado de líneas, lo que significa que el objeto cargado no tiene aplicado ningún mapeado. Antes de generarlo nosotros con el programa, vamos a optimizar el modelo.
Seleccionamos todos los polígonos en *Tools / Select / All* (el entramado de líneas se volverá de color rojo), seguidamente elegimos la opción *Tools / Optimize models*. Una vez optimizado, procederemos a realizar el mapeado UV. Como el gusano tiene forma de tubo, es suficiente con generar un mapa lateral del mismo, para ello elegimos la opción *Tools / UV Mapping* y seleccionamos el modo "Planar", y aplicamos la opción "Y - Top" en "Positive". El entramado se transformará y mostrará una vista lateral de todo el *Slunk*. Para terminar exportamos de nuevo el modelo en *File / Model / Save* en formato .OBJ con el nombre *Slunk* en el mismo directorio para pasarlo al Deep Paint 3D. (Fig. 9).

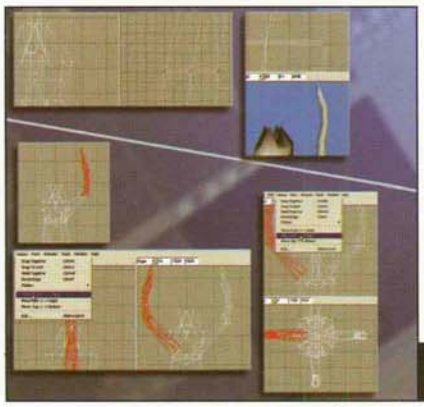
TEXTURIZANDO CON DEEP PAINT 3D

Al cargar nuestro modelo, observamos que en la ventana



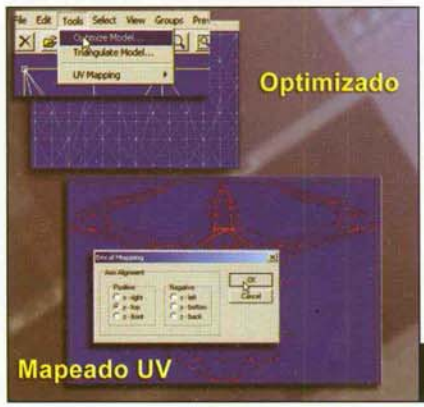
7

A partir de cilindros es posible obtener cualquier forma desplazando vértices.



8

Copiando y realizando "mirrors" podemos crear y colocar el resto de las patas.

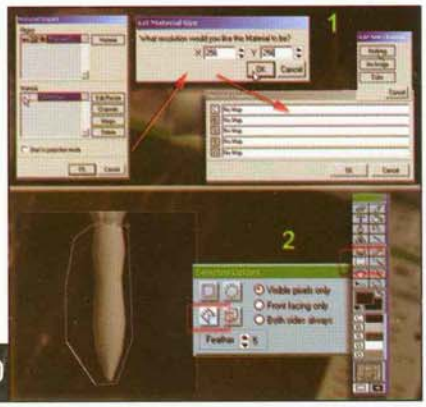


9

Con LithUnwrap podemos optimizar el modelo antes de crearle el Mapeado UV.

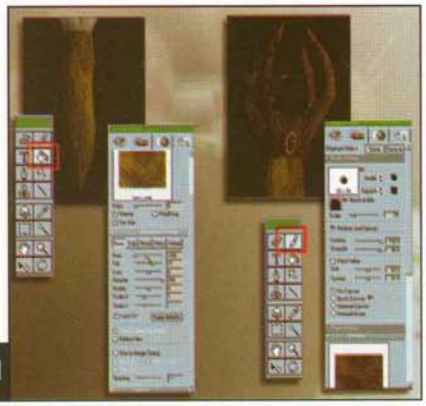
NOTA

En el directorio "Extras" del CD se encuentran todos los modelos explicados en esta sección.



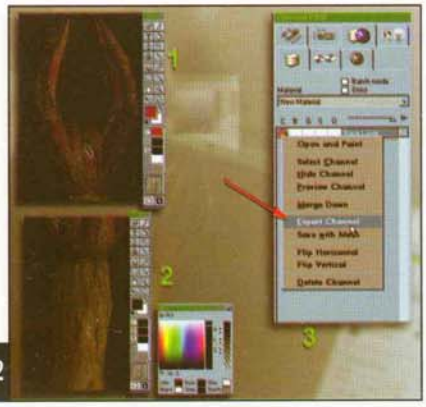
10

Podemos rellenar partes de un modelo seleccionándolo con anterioridad.



11

Mediante las herramientas de relleno y pincel podemos pintar nuestro modelo fácilmente.



12

Una vez acabado el texturizado podemos grabar la textura con la opción Export Channel en canal de color.

TRUCO

Podemos rotar el modelo de una manera más rápida, dejando pulsada la tecla "R" y desplazando el ratón con el botón izquierdo pulsado.

"Material Import" se indica la presencia de un mapeado UV. Debemos entonces crear un material vacío para poder continuar. Así que pinchamos en el primer cuadro de la sección "Material" en "Unknown" y elegimos un tamaño de 256 x 256 y como nuevo canal de color ("C") seleccionamos en "Add New Channel" "Nothing". Seguimos, seleccionando la vista "Top" en "View". Seguidamente ajustamos el modelo en la ventana y procedemos a pintarlo (Fig. 10).

Primeramente vamos a rellenar con una textura el cuerpo del gusano, para ello, lo seleccionamos con la herramienta de selección por líneas. Es necesario ir rotando el modelo y sumando selección, con la tecla "Shift" pulsada, para elegir todo las vistas del cuerpo.

Una vez seleccionado por completo el cuerpo elegimos la textura "Skin 3" (en la sección *Texture Paints* del *Command Panel*) y le modificamos el tono (Hue –248) y la saturación de color (Sat 77) en la pestaña *Base* en *Advanced Behaviour*. A continuación, rellenamos con la herramienta de relleno. Una vez pintado el cuerpo, pasaremos a la cabeza y patas. Eliminamos la selección anterior con "Ctrl" + "D" y elegimos la herramienta del pincel. Escogemos la textura "Elephant Hide +" en *Skins, Fur and Hair*. Luego, escalamos el pincel a un 25 % en la sección *Brush Settings* en "Scale" y pintamos sin pasar por encima del mismo sitio muchas veces (Fig.11).

Siempre es más profesional añadir detalles a la textura como manchas de sangre o cierta suciedad pegada en la piel. Para ello, sólo tenemos que pintar. Las machas de sangre las haremos con una variación del pincel ("Brush"). En "Variations" *(Command Panel)* elegimos la variedad "Pastel Oil +", la cual nos dará un trazo suave y diluido. Escalamos el pincel al 15%, seleccionamos una tonalidad de color rojo y pintamos sobre las mandíbulas y las patas, Para la suciedad elegimos la misma herramienta y variación de pincel, pero cambiamos a una tonalidad oscura y verdosa. Pintamos en la unión de la cabeza y el tronco y líneas sobre el tronco.

Una vez acabado, sólo nos queda exportar el material en el directorio "C:\juego_ZOF\slunk\texturizado\" con el nombre "textura_slunk.bmp". Para ello, elegimos la opción *Export Channel* del menú emergente que aparece al pulsar el botón derecho del ratón sobre el canal de color "C" en materiales.

MODELADO DE UNA PLANTA LUNY

El Modelado de un *Luny* es extremadamente sencillo. Sabemos que esta planta posee movimiento. Como éste lo aplicaremos en CharacterFx, tenemos que realizar el modelado en una postura rígida y erguida. Simplemente partimos de una esfera de 6 "Stacks" y 12 "Slices". Seleccionamos los vértices de la base, con *Ignore BackFaces* deseleccionado para escoger todo el perímetro, y los escalamos hacia fuera con F4 en la vista "Top". A continuación, hacemos lo mismo con los vértices centrales, hasta obtener la forma de una campana. Para terminar de dar forma debemos crear la boca, seleccionando las dos filas superiores de vértices y desplazándolas hacia el interior de la pieza.

Guardamos el modelo en el directorio "C:\juego_ZOF\modelado\luny\" con el nombre "luny.ms3d".

En el próximo número...

... terminaremos nuestro *Luny*, además de crear los *Shaarks*. También, seguiremos creando los elementos de "Zone of Fighters", en esta ocasión toca modelar y texturizar las plantas carnívoras.

Herramientas para hacer la música de un juego

Como ya sabemos, el complemento imprescindible para proporcionar audio a un juego es la música.

Hay multitud de sistemas para realizarla utilizando nuestro ordenador. Hasta ahora, hemos realizado efectos de sonidos aislados con editores de audio especializados como GoldWave, Sound Forge o Cool Edit Pro. Pero para hacer música necesitaremos algo más que simples efectos aislados. Vimos que la utilización de secuenciadores como Cubase VST nos permitía mezclar y sincronizar eventos musicales MIDI y muestras de audio para generar un tema musical.

UTILIZACIÓN DE CAJAS DE RITMOS

Existe otro tipo de secuenciadores muy utilizados hoy día que permiten unir muestras en forma de notas musicales, ritmos o voces para crear música. Son las cajas de ritmos virtuales. Actualmente, es muy común este tipo de aplicaciones para la creación de música de baile y juegos de ordenador ya que son de fácil uso y proporcionan un resultado excelente.
Generalmente, son muy usadas para realizar ritmos o loops de muestras para incluir en un tema musical mayor, normalmente realizado en un secuenciador MIDI y a través de instrumentos externos. Hay infinidad de ellas, pero sólo explicaremos el uso básico de las dos cajas de ritmos más utilizadas y potentes: *Rebirth* y *Fruity Loops*. En ambas aplicaciones vamos a realizar un sencillo loop que posteriormente exportaremos en formato .WAV para poder insertarlo en nuestro secuenciador favorito.

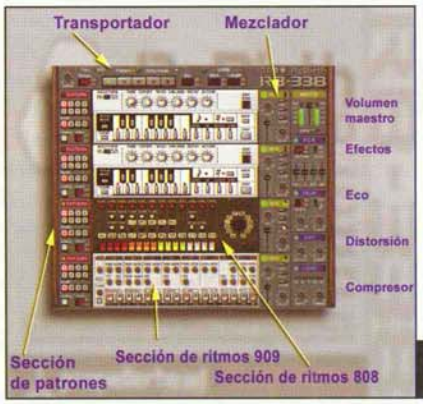
CREANDO UN LOOP CON "REBIRTH RB-338"

Rebirth- RB-338 nace de la idea de *Propellerhead* (Steinberg) de simular en un ordenador los módulos de sonido *TB 303, TR 808* y *TR 909* de *Roland*. Su manejo se basa en dos modos de trabajo: patrones y canciones. Los patrones nos servirán para crear canciones enteras, las cuales se podrán grabar en disco desde el compás indicado por "Start" hasta el indicado por "Lenght" con el modo "Loop" activado. En la figura 1 se muestra una descripción de los diferentes módulos de la interfaz del programa.

A continuación, vamos a construir un loop que luego exportaremos.

En primer lugar, crearemos patrones aprovechando las cuatro secciones de que disponemos. Trasladaremos estos patrones al modo "Song", en donde crearemos 12 compases para formar el loop definitivo. Una vez en el modo "Song" es posible modificar el patrón en tiempo real, cambiando parámetros o mutando secciones. Conmutamos a la sección "Pattern" para preparar nuestros patrones. Por defecto, un patrón está formado por 4 compases y un *tempo* de 140. Y en cada sección podemos disponer de 4 bancos (A,B,C,D) con 8 patrones cada uno. Así que en la sección de patrones pinchamos sobre el número 1 (patrón 1). La base rítmica la haremos con las secciones 808 y 909. Nos situamos en la sección 909 y hacemos clic sobre el primer botón de cada compás como se muestra en la figura 2.

Para seguir la base rítmica añadiremos algunos golpes de percusión en la sección 808 y subimos el volumen de ésta en su mezclador. Una vez que tenemos



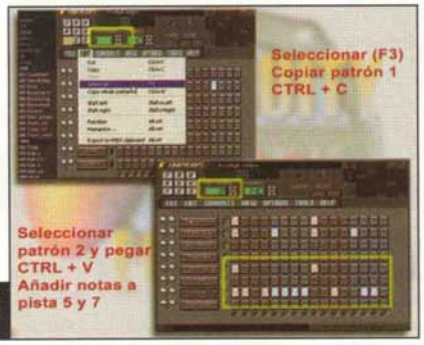
Descripción básica de las partes del programa Rebirth-338.

El primer paso es asignar las notas en las secciones de ritmos 808 y 909 en el modo "Pattern".

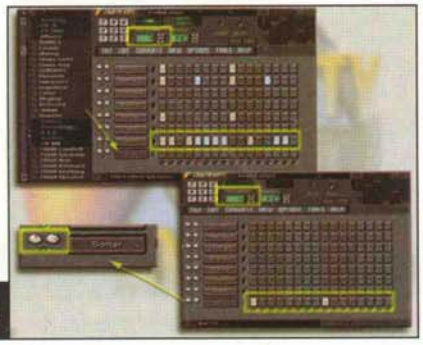
En el modo de grabación podemos activar o desactivar secciones, así como cambiar parámetros en tiempo real.

4 *Descripción básica de las partes de Fruity Loops.*

5 *Es posible asignar cualquier muestra de las librerías de samplers a una pista (arriba). En la parte inferior se muestran las notas escritas en las pistas 1 y 2.*



6 *Podemos copiar patrones enteros con sólo las acciones de copiar y pegar.*



7 *Notas aplicadas a las pistas 5, 7 y 8.*

la base rítmica, añadiremos los patrones de las secciones de sintetizador como se muestra igualmente en la figura 2.

Terminado de construir el patrón principal, pasamos al modo "Song", colocamos un 1 en "Start" y 14 en "Lenght". De esta forma creamos una canción de 14 compases. A continuación, debemos inicializar la canción con el nuevo patrón creado en el modo "Pattern". Para ello elegimos la opción *Edit / Initialize Song from Pattern Mode*. Mientras se graba, Rebirth permite modificar los parámetros, así como cambiar patrones o mutar las distintas secciones. Utilizaremos esta posibilidad para activar y desactivar las secciones a lo largo de los 14 compases. (Para tener una referencia, en "Extras" del CD se encuentra el loop terminado "loop_rebirth.wav"). Para finalizar, ya podemos exportar la canción como un loop con la opción *File / Export Loop as Audio File* (Fig. 3).

CREANDO UN LOOP CON FRUITY LOOPS

Fruity Loops es una estupenda herramienta para construir loops y canciones y exportarlas en formato .WAV o .MIDI. Posee un sintetizador de bajos integral, hasta 8 efectos simultáneos y además puede importar ficheros .WAV o .SYNC para crear librerías de sonidos. Vamos a explicar de una manera básica los pasos para crear un loop y exportarlo en formato .WAV.

En la figura 4 se muestra una pequeña descripción de las partes del programa.

Antes de comenzar a crear nuestros patrones, vamos a preparar los sonidos de cada pista. Por defecto, se visualizan un total de 8 pistas con 4 compases por cada una de ellas y un *tempo* de 140. Cambiemos el sonido de la primera pista. Para ello, abramos la librería de muestras de la "TR 808" haciendo clic sobre ella situada en la lista de samples. Sin dejar de pulsar el botón izquierdo del ratón, desplazar la muestra "TR808 Snr_snap" al botón de la pista primera. El sonido de esta pista cambiará. Pulsando el botón de cada pista podemos editar cada una de las muestras que contiene, abriéndose la ventana "Channel Settings". Haremos la misma operación para la pista 2, pero eligiendo la muestra "TR909 Klick" de la librería de la "TR 909" (Fig. 5).

Construyamos, a continuación, los patrones que nos servirán para formar el loop. Empezamos, pulsando el primer botón de cada compás en la segunda pista y el primero de los compases 1 y 3 de la primera. Seguidamente, crearemos otro patrón a partir del primero, pero añadiendo notas. Para ello, nos situamos en el primer patrón, lo seleccionamos por completo en *Edit / Select All* o pulsando "F3", y lo copiamos en memoria con "Ctrl" + "C". A continuación, seleccionamos el patrón 2 a través del selector de patrones o en la casilla "Pattern" y copiamos el patrón primero con "Ctrl" + "V". En el patrón 2 añadiremos más notas en las pistas 5 y 7 como se muestra en la figura 6.

Realizamos la misma operación anterior, copiando el patrón 2 en el 3 y añadiendo más notas en la pista 8, a la que previamente le hemos cambiado el sonido por la muestra "Sonar" de la librería "Effects". Además, a la pista 8 le subimos al máximo el volumen. Para terminar, creamos un cuarto patrón con sólo dos notas en la pista 8 al principio del compás primero y tercero. Bien, ya hemos construido los 4 patrones que utilizaremos para componer la canción que servirá de loop definitivo (Fig. 7).

Tanto Rebirth como Fruity Loops son aplicaciones que permiten un gran abanico de posibilidades para la creación de loops.

En el próximo número...

... terminaremos nuestro loops en Fruity Loops y seguiremos estudiando algunas características más de esta herramienta.

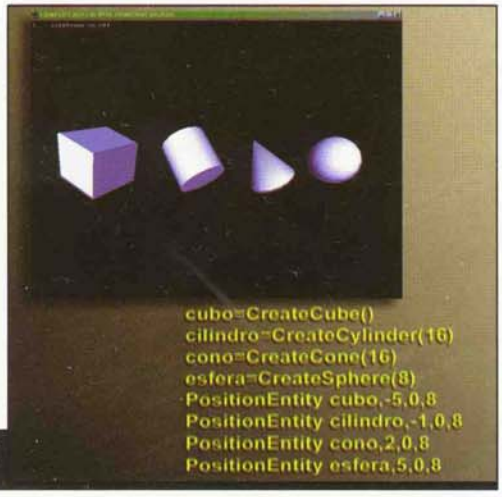
Manejo de funciones 3D (I). Objetos 3d

Para estudiar las funciones 3D que Blitz3D posee es necesario empezar por la creación y manejo de objetos 3D comúnmente denominados "mesh".

Como ya hemos visto a lo largo del curso, en Blitz3D es posible cargar y manipular *meshes* realizados con cualquier programa de modelado 3D. Pero también es posible crearlos, desde la nada, mediante programación y en tiempo real. Aprenderemos cómo utilizar ambos sistemas de una forma práctica y conoceremos los trucos que nos ayudarán a conseguir asombrosos efectos para nuestros juegos.

CREANDO PRIMITIVAS

Blitz3D permite la creación de objetos 3D básicos o primitivas. Podemos crear directamente: cubos, esferas, cilindros y conos. Para cada uno de ellos se utiliza un comando diferente:



1

Blitz3D permite la creación de primitivas básicas con una sola instrucción.

Cubo:

```
CreateCube ([entidad madre])
```

Esfera:

```
CreateSphere([nº de segmentos]
[,entidad madre])
```

El nº de segmentos indicará más o menos polígonos para crear la esfera. Por defecto, su valor es 8; es decir, 224 polígonos, aunque es posible utilizar rangos desde 2 hasta 100.

Cilindro:

```
CreateCylinder([nº de segmentos]
[,sólido true/false]
[,entidad madre])
```

Si en la opción "sólido" colocamos "true" se creará un cilindro sólido y si colocamos "false" crearemos un tubo hueco. El rango posible en el nº de segmentos varía desde 3 hasta 100 inclusive.

Cono:

```
CreateCone([nº de segmentos]
[,sólido true/false]
[,entidad madre])
```

Las opciones son exactamente iguales que en el comando anterior. La única diferencia es que el parámetro "sólido" indica si el cono tiene o no base (Ver "ejemplo1.bb").

CARGANDO OBJETOS 3D

Generalmente, necesitaremos utilizar en nuestros juegos objetos que hayamos creado previamente con cualquier herramienta de modelado. Blitz3D posee un par de comandos que posibilitan la importación de objetos desde un archivo, en formato .X, .3DS o .MD2 (además del formato .B3D a partir de la versión 1.76): "LoadMesh" y "LoadAnimMesh". Vamos a estudiar cómo usar cada uno de estos comandos, ya que es muy importante tener algunos conceptos claros si queremos manipular modelos formados por una sola parte, por varias o con animación.

USO DE LOADMESH

Esta función nos permitirá importar un modelo 3D. El objeto cargado será convertido a un solo *mesh*; es decir, si cargamos, por ejemplo, el modelo de una persona, donde la cabeza, el tronco y las extremidades fueran partes diferentes, esta función no diferenciaría esas partes. Lo mismo ocurrirá si el modelo tuviese algún tipo de animación. De todas formas, tampoco resulta conveniente utilizar "LoadMesh" ya que también bajaría el rendimiento del programa. Así que esta función es ideal para cargar objetos estáticos formados por una sola parte, como una piedra o una pared.

```
LoadMesh (nombre fichero$
[,entidad madre])
```

LOADANIMMESH

Por otro lado, esta función nos permitirá importar modelos con algún tipo de animación. Además, es lo que debemos usar si queremos cargar un objeto que esté formado por varias partes, como por ejemplo, la cabeza, tronco y extremidades en el modelo de un hombre. Luego, mediante las funciones de manejo de "Children" ("CountChildren" o "GetChild")

2

Mediante programación es posible crear objetos 3D en tiempo real.

podremos texturizar cada parte o moverlas independientemente. Pero de estos procedimientos hablaremos después. Antes, vamos a continuar estudiando otros métodos de creación de *meshes*.

```
LoadAnimMesh (nombre fichero$
[,entidad madre])
```

CREANDO Y MODIFICANDO TUS PROPIOS *MESHES*. VÉRTICES, TRIÁNGULOS Y SUPERFICIES

Por medio de la programación es posible crear también cualquier tipo de objeto 3D más o menos simple; es decir, vamos a utilizar funciones de Blitz3D para modelar nuestros *meshes*. Pero antes, debemos aprender algunos conceptos básicos sobre cómo Blitz3D trabaja los objetos 3D.

Realmente, un *mesh* no está formado por vértices y triángulos, sino por superficies *(surfaces)*. Y son éstas las que están formadas por vértices y triángulos. Deducimos entonces que para tener una superficie es necesario al menos un triángulo y para formar un *mesh*, una superficie. Podemos, además, controlar el dibujado o renderizado de los triángulos de una superficie por medio de los "Brushes", con lo que podemos, por ejemplo, aplicar a cada triángulo propiedades como: pintar con un color, aplicar texturas o dar más brillo. Pero esta opción la veremos en el próximo número.

La forma que veremos para crear primitivas es similar a como lo haría una aplicación de modelado, la única diferencia es que ésta utiliza una interfaz gráfica y nosotros, programación. Vamos a crear un objeto 3D uniendo vértices y triángulos. Para empezar, es necesario crear un *mesh* nuevo con la función "CreateMesh". Para que este *mesh* tenga alguna forma es necesario crearle una superficie con la función "CreateSurface". A partir de ahora, ya estamos preparados para sumar vértices a esta superficie nueva con la función "AddVertex". Una vez creados los vértices debemos unirlos para formar los triángulos, los cuales darán la forma definitiva al *mesh*. Realizaremos esta operación con la función "AddTriangle". Para terminar el proceso debemos actualizar las normales del nuevo mesh, este procedimiento es imprescindible si queremos que se ilumine correctamente. Para ello aplicamos la función "UpdateNormals" (Ver "ejemplo2.bb").

Seguidamente, se muestran los pasos para crear el lado de un cuadrado utilizando estos procedimientos:

```
nuevo_Mesh=CreateMesh()
Superficie=CreateSurface(nuevo_Mesh)
AddVertex  Superficie, -1, 1, 0
AddVertex  Superficie, 1, 1, 0
AddVertex  Superficie, 1, -1, 0
AddVertex  Superficie, -1, -1, 0
AddTriangle Superficie, 0, 1, 2
AddTriangle Superficie, 0, 2, 3
UpdateNormals nuevo_Mesh
```

También es posible modificar nuestros propios *meshes* en tiempo real cambiando el color, la textura o la posición de sus vértices (Ver "ejemplo 3.bb").

3

Con "VertexCoords" podemos modificar las coordenadas de cada vértice posibilitando la deformación de superficies.

Si una vez creado el *mesh*, queremos modificar las coordenadas de los vértices de las superficies que lo componen debemos utilizar el comando "VertexCoords":

```
VertexCoords Superficie,
Nº del Vértice, Posición X#
del Vértice, Posición Y#,
Posición Z#
```

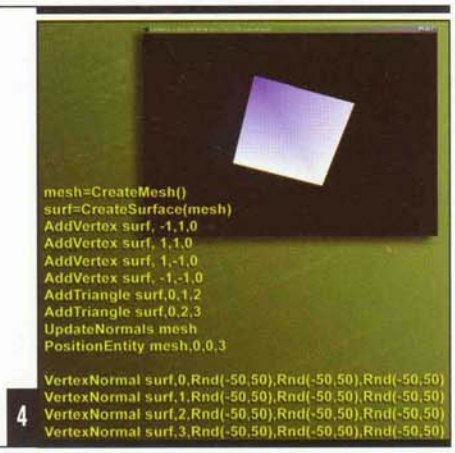
Si la superficie estuviera texturizada, también podemos cambiar las coordenadas de la misma mediante la instrucción "VertexTexCoords":

```
VertexTexCoords  Superficie,
Nº del Vértice, Coordenada U#
del vértice, Coord.. V#
[,coord.. W#]
```

U#, V# y W# son el equivalente a X#, Y# y Z# pero referidos al mapeado de textura.

Mediante la instrucción "VertexNormal" es posible cambiar la normal de un vértice; es decir, la dirección de éste con referencia a las coordenadas X, Y y Z:

```
VertexNormal Superficie,
Nº del Vértice, NX# (Normal
de la coordenada X del
vértice), NY#, NZ#
```



4

Modificando la normal de un vértice es posible cambiar su orientación con respecto al resto de los vértices de una superficie.

Por último, también es posible cambiar el color de cada vértice con la instrucción "VertexColor", por lo que conseguir superficies multicolores es tarea sencilla (Ver "ejemplo4.bb"):

```
VertexColor, Superficie,
Nº del Vértice, Componente
de Rojo#, Verde#, Azul#
```

Además, es posible pintar una superficie independiente con un determinado *brush* utilizando la función "PaintSurface":

```
PaintSurface Superficie,
Pincel (brush)
```

Es muy común, en muchos juegos, la utilización del sistema *LOD* (nivel de detalle) para la representación de los objetos 3D. Blitz3D sólo posee está técnica para la representación de terrenos, pero es posible realizar una función que lo haga también con los objetos. El fundamento principal de esta técnica es reducir polígonos; es decir, borrar superficies para luego crearlas de nuevo con menos vértices y triángulos, manteniendo siempre la forma original lo más aproximada posible. Para facilitarnos esta labor, Blitz3D nos suministra una función importante: "ClearSurface".

```
ClearSurface Superficie,
[Borrar todos los vértices
(true / false)][,Borrar todos
los triángulos (true / false]
```

Esta función nos permitirá borrar partes o superficies de un *mesh* inmediatamente.

MANIPULANDO OBJETOS 3D

Además de transformar en tiempo real cada vértice de la/s superficie/s de un *mesh*, podemos desplazar, rotar, escalar, pintar o iluminar un *mesh* completo, de igual forma como lo hacemos con una entidad. Para posicionar un *mesh* en el mundo 3D se utiliza el comando "PositionMesh":

```
PositionMesh Variable que
manipula el Mesh, X#, Y#, Z#
```

Aplicarle una rotación en cualquiera de sus 3 ejes es sencillo utilizando "RotateMesh":

```
RotateMesh Mesh, Pitch#(eje X),
Yaw# (eje Y), Roll# (eje Z)
```

También es posible cambiar el tamaño de cualquier objeto 3D con la instrucción "ScaleMesh":

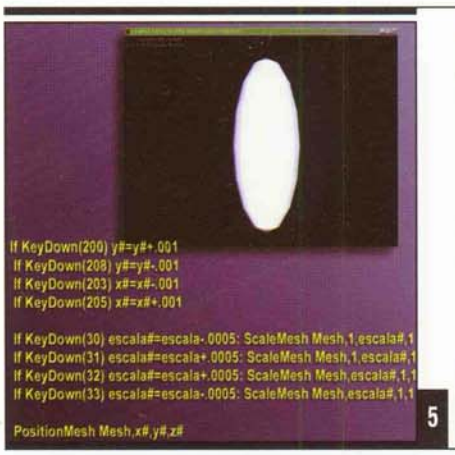
```
ScaleMesh Mesh, Escalado eje X#,
Escalado Y#, Escalado Z#
```

(Ver "ejemplo5.bb").

Si deseamos aplicar un escalado determinado uniforme a nuestro modelo debemos utilizar la instrucción "FitMesh". Ésta, además de escalar, realiza una traslación de los vértices del modelo:

```
FitMesh Mesh, Posicion X# del
mesh, Posición Y, Posición Z,
Longitud nueva del Mesh#,
Altura nueva del Mesh#,
Anchura nueva del Mesh#
[,Distorsion]
```

El parámetro de distorsión, por defecto en *false*, nos sirve para indicarle al Blitz3D que el *mesh* se ajustará uniformemente a la nueva medida; es decir, aumentar por igual el tamaño en los tres ejes.



5

Independientemente de las funciones de modificación de entidades es posible utilizar funciones propias para los mesh como "PositionMesh".

También podemos ahorrarnos un montón de instrucciones si deseamos cambiar el aspecto de un *mesh*, ya que es posible pintarlo con un *brush* (pincel) previamente definido utilizando la instrucción "PaintMesh":

```
PaintMesh Mesh, Brush
```

Y por último, una interesante instrucción que nos permitirá iluminar un *mesh* completo o una porción de él; "LightMesh":

```
LightMesh Mesh, Componente de
color Rojo#, Verde#, Azul#
[,Rango de la luz], Posición X
de la luz#, Posición Y#,
Posición Z#
```

OBTENIENDO INFORMACIÓN DE LOS *MESHES*

Es muy común la necesidad de obtener cierta información de los modelos que utilizamos en nuestro juego como su tamaño, la cantidad de superficies que lo forman, etc.

Por ejemplo, queremos asignar un sistema de colisión por caja a un objeto 3D (ver sistema de colisión "EntityBox"). Para saber exactamente qué tamaño tendrá esa caja debemos saber el

tamaño del modelo. Para ello, disponemos de las instrucciones "MeshWidth", "MeshHeight" y "MeshDepth":

```
Longitud_Mesh = MeshWidth#
( Mesh )    ; X
Altura_Mesh = MeshHeight#
( Mesh )    ; Y
Anchura_Mesh = MeshDepth#
( Mesh )    ; Z
```

En ocasiones, puede ocurrir que necesitemos aislar una o varias superficies del conjunto que forma un *mesh*, para desplazarlas, cambiarlas de color, etc. Un buen ejemplo podría ser la implementación de daños en un coche de carreras, desplazando superficies de la estructura para crear abolladuras o rompiendo cristales. Existen un par de comandos que nos ayudarán a realizar esta operación: "CountSurfaces" y "GetSurfaces".

```
GetSurface (mesh, nº de superficie)
```

Mediante este comando obtenemos la superficie determinada por el parámetro "nº de superficie" en una variable, la cual servirá para manipular dicha superficie. Pero antes, es necesario recorrer en un bucle cada una de las superficies del modelo desde 1 hasta la última de ellas determinada por la función "CountSurfaces":

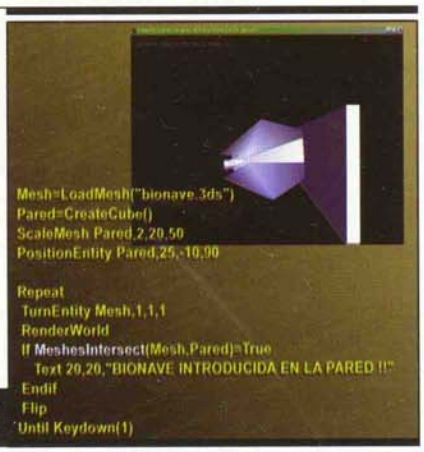
```
CountSurfaces (mesh)
```

Para terminar, una función realmente interesante y muy útil: "MeshesIntersect".

A veces, en muchos juegos se puede observar cómo parte del protagonista u otro personaje se introduce en la pared cuando se mueve. Este efecto queda realmente feo. Para evitar este problema podemos utilizar esta función. Lo que hace es simplemente avisar de cuándo dos objetos están mezclándose:

```
MeshesIntersect (Mesh A, Mesh B)
...
If MeshesIntersect ( Alien,
Pared) = True    Text "El
Alien se mezcla con la pared"
...
```

No es conveniente su utilización masiva ya que no es muy rápida y puede bajar el rendimiento del juego (Ver "Ejemplo6.bb").



6

La función "MeshesIntersect" nos permite controlar si dos modelos se están mezclando.

UTILIZACIÓN DE "CHILD"

Supongamos que hemos modelado un hombre con Milkshape 3D y salvamos el modelo sin agrupar, ya que la cabeza, el tronco y las extremidades son partes diferentes. Al no agrupar, el modelo estará formado por varios *meshes*. Estas partes son tratadas en Blitz3D como hijos (Children). Esto nos ayudará si queremos texturizar o mover cada parte independientemente. Además, cada una de ellas puede ser tratada como una entidad diferente con todo el poder que ello supone.

Podemos saber cuántas partes tiene un modelo contando los hijos del mismo. Para ello utilizaremos la función *CountChildren (Entidad)*

7

Mediante el manejo de "Children" es posible controlar las partes que forman un modelo.

Incluso si hemos nombrado cada parte en MilkShape 3D (u otra aplicación), es posible preguntar desde Blitz3D por ese nombre para encontrar esa parte (Child) utilizando la función:

```
FindChild (Entidad, nombre
de la parte$)
```

También podemos obtener una parte por su posición en la jerarquía del modelo mediante la función *GetChild (Entidad, número de la parte)*.

Si deseamos, por ejemplo, modificar la posición de todas las partes de un modelo, debemos utilizar un bucle que recorra cada una de ellas (Ver Fig. 7):

```
Hombre = LoadAnimMesh ("hombre.3ds")
For n = 1 to CountChildren (Hombre)
   Parte = GetChild (Hombre , n)
   MoveEntity Parte, 1, 1, 1
Next
```

Hasta aquí nuestro primer estudio de las funciones 3D para el manejo de *meshes*.

En el próximo número...

... nos dedicaremos al texturizado de objetos y a explicar los *brushes*.

Editores de audio. Cool Edit Pro 2.0 (II)

Vamos a terminar esta serie de tutoriales sobre editores de audio aprendiendo a trabajar con el multipista de Cool Edit Pro 2.

Podemos observar que el *modo multipista* es básicamente igual al *modo de edición*, la diferencia radica en que es posible trabajar con varios sonidos a la vez y mezclarlos. La mejor forma de aprender es practicando, así que vamos a realizar la mezcla de audio para crear el MP3 que sirve de argumento para nuestro juego "Zone of Fighters".

PREPARANDO LOS SONIDOS ANTES DE MEZCLAR

En el directorio "Extras" del CD disponemos de dos ficheros .WAV . Uno de ellos corresponde a la música que utilizaremos de fondo y el otro es una grabación en bruto de una voz en off narrando el argumento del juego. Nuestra misión es preparar esa voz para luego mezclarla con la música en el multipista.

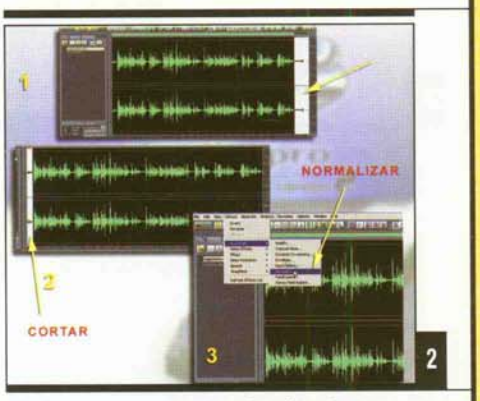
Una vez cargado el programa, nos encontramos en la ventana principal de edición, la cual explicamos en el número anterior de este tutorial. Pulsando en el icono pasamos al *modo multipista*. Allí vamos a insertar en la primera pista el audio correspondiente a la música que irá de fondo. Nos situamos sobre la primera pista vacía y pulsamos el botón derecho del ratón. Aparecerá un menú flotante en el que elegiremos la opción *Insert /Wave from File*. Elegimos el fichero contenido en el directorio "Extras" del CD llamado "musica.wav". Inmeditamente, aparece un bloque de audio en la pista primera conteniendo la música (Fig. 1).

Podemos insertar ficheros de audio directamente en las pistas en la ventana multitrack.

Vamos a continuación a insertar la voz en la segunda pista. Así que pulsamos sobre ella. Pero antes de seguir, es preciso preparar la voz en el *modo edición*, ya que está en bruto. Pulsamos en el botón para pasar al *modo edición*. Una vez situados en él cargamos del CD el fichero llamado "argumento.wav", el cual vamos a preparar para mezclar posteriormente con la música. Una vez cargado, vamos a eliminar el audio sobrante del principio y fin, seleccionando las partes a borrar y pulsando "Ctrl" + "X" o pulsando la tecla "Suprimir". A continuación, y como viene siendo habitual, realizaremos un normalizado del volumen al 100 % mediante la opción *Effects / Amplitude/Normalize* (Fig. 2).

Seguidamente, aplicaremos un efecto de reverberación o eco para simular la inmensidad del espacio exterior. Para ello, elegimos el efecto "Reverbs" situado en *Effects/Delay effects*. Elegimos el preset *Concert Hall Light* y cambiamos los siguientes parámetros del efecto:

- **Total Reverb Length** (Longitud total del reverb) a 1190 ms.
- **Attack Time** (Tiempo de ataque) a 240 ms.



El primer proceso es cortar el audio sobrante y normalizar el volumen.

- **High Frequency Absorption Time** (Tiempo en eliminar las altas frecuencias) a 630 ms.
- **Perception** (cantidad percepción, con suavidad o eco) a 30.

A continuación, también cambiamos los parámetros de mezcla del efecto:

- **Original Signal (Dry)** (Porcentaje de señal original) al 70 %.
- **Reverb (Wet)** (Cantidad de reverberación) un 165 %.

MEZCLANDO TODO EN EL MULTIPISTA

Una vez aplicado el efecto, ya tenemos la voz preparada para pasarla al multipista y mezclarla con la música. Debemos entonces insertar el sonido a la ventana *Multitrack*, eligiendo la opción *Insert in Multitrack* en *Edit* o pulsando "Ctrl" + "M" (Fig. 3). De nuevo, pasamos al modo multipista y observamos cómo en la pista segunda se ha insertado correctamente un nuevo bloque conteniendo el audio de la voz ya preparado.

Para desplazar el bloque de la voz a lo largo de la pista, por ejemplo, para ajustar su reproducción en un determinado lugar, nos situamos sobre ella y move-

Una vez terminado el tratamiento del sonido podemos insertarlo en el multipista con "CTRL" + "M".

mos el ratón sin dejar de pulsar el botón derecho. Una vez situados los bloques de audio debemos ajustar los volúmenes de cada pista para conseguir una mezcla perfecta.

En primer lugar, vamos realizar una subida rápida de volumen al comienzo del bloque de voz, para que su entrada en escena se produzca de forma suave. Para ello, debemos pulsar en el botón para visualizar en el bloque la gráfica de la envolvente del sonido, es decir, el volumen. En un principio, se muestran dos cuadrados blancos, al principio y al final de la onda, unidos por una línea. Esta línea representa el nivel de volumen del sonido. Pulsando sobre cada cuadrado podemos desplazarlos y así cambiar el volumen en ese punto. Para realizar, por ejemplo, subidas o bajadas de volumen, podemos crear todos los puntos (cuadrados) de inflexión que queramos con sólo pulsar con el ratón sobre la línea. De momento, nos interesa una subida rápida al principio. Solamente tenemos que crear un punto nuevo a continuación del primero. Este punto indicará el tope del volumen en la subida y el primero el comienzo.

Hay una forma más rápida de aumentar el volumen general de una pista y es a través del vúmetro que aparece al pulsar con el botón derecho sobre "V0" □. Vamos a continuar preparando los cambios de volumen en la pista de la música. Para ello, realizaremos la misma operación anterior, pero en esta ocasión añadiremos más puntos de inflexión para crear diferentes cambios de volumen dependiendo de la zona que esté sonando. Bajaremos entonces el volumen cuando esté reproduciéndose la voz (para mantener de fondo la música) y lo subiremos cuando ésta haya acabado.

Podemos también añadir cambios en el panorámico del sonido de la misma forma que hicimos con el volumen, a través de una gráfica y puntos de inflexión. Para visualizar esta gráfica pulsamos sobre el botón . A continuación, realizaremos los cambios en el panorámico sumando puntos de inflexión y desplazándolos.

También podemos cambiar el panorámico de una pista de manera general pulsando sobre "Pan0" con el botón derecho del ratón.

OTRAS OPCIONES

A través del multipista podemos cambiar los parámetros de un efecto aplicado a un sonido insertado en una pista. Además, también es posible añadirle efectos nuevos.

Para editar un efecto, es imprescindible que el sonido ya tenga uno asignado desde el modo *edición*. En nuestro caso, recordemos que en el audio de la voz en off habíamos aplicado una reverberación. Pues bien, para editar desde el multipista ese efecto, sólo tenemos que pulsar sobre el botón "FX2" . Aparecerá entonces un panel gráfico con todos los parámetros del efecto.
Además, podemos mezclar si queremos que suene más o menos reverberación pulsando en la pestaña "Mixer" en ese mismo panel.

CONVIRTIENDO TODAS LAS PISTAS EN UN SOLO FICHERO DE AUDIO

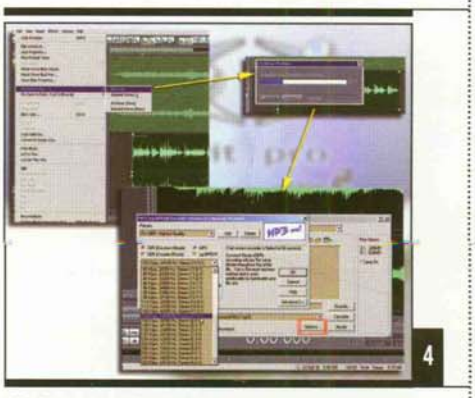
Una vez terminada la mezcla, necesitaremos obtener un solo fichero de audio para que pueda ser incluido en nuestro juego. Debemos entonces elegir la opción *Mix Down to File* en el menú *Edit* del multipista y elegir la opción *All waves*. Con esto, le estamos diciendo a Cool Edit Pro 2 que nos mezcle todas las pistas de audio en una sola. Cuando el programa termina el proceso de mezclado, pasa directamente a la ventana de edición y nos muestra la onda resultante del proceso. (Fig. 4).

Seguidamente, podemos normalizar de nuevo el volumen para tener definitivamente preparado el argumento del juego listo para grabar en formato .MP3.

GRABANDO EN .MP3

Para salvar el audio en MP3 elegimos la opción *File / Save as*. En el apartado *Tipo* elegimos el formato *mp3PRO(Fhg)(*.mp3)*. Después, elegimos la calidad resultante pulsando en el botón "Options" y elegiendo la opción *128 Kbps, 44100 Hz, Stereo (11.0:1)*.

En este número hemos conocido la enorme utilidad del *modo multipista* de Cool Edit Pro 2. Ponemos fin aquí a la serie de tutoriales que hemos dedicado a los editores de audio profesionales más utilizados.



Mediante la gráfica de la envolvente podemos modificar el volumen de la pista en cualquier lugar.

En el próximo número...

... comenzamos una serie dedicada al desarrollo y programación para juegos. Empezaremos por los mundos BSP en Blitz3D.

Juegos de estrategia (I). Estrategia por turnos

Empezamos una serie de dos números dedicados a los juegos de estrategia. Miraremos atrás en la Historia para conocer los antecedentes que han servido para crear la estrategia en nuestro PC.

La clasificación de este género se reduce, básicamente, a juegos de estrategia por turnos y estrategia en tiempo real.

DEL TABLERO AL ORDENADOR

La estrategia siempre ha estado ligada al arte de la guerra. Este tipo de juegos ha existido a lo largo de la Historia en forma de tableros. Quizás el más popular y antiguo sea el ajedrez. Pero como estrategia bélica jugada sobre un tablero fueron *Estratego* y *Warhammer* los que ganaron popularidad a partir de los años 40. El primero se basaba en las guerras en Europa, mientras que el segundo se basaba en mundos de fantasía repletas de batallas entre humanos y orcos. Éstos y otros juegos de tablero tenían un denominador común: el uso de los dados para determinar el rumbo de una partida. Con la llegada de los ordenadores personales, todos estos títulos, que al principio llenaban una mesa entera de fichas, cartas y figuras en miniatura de las tropas, pasaron a la pantalla. El azar de los dados se transforma en decisiones lógicas del ordenador tomadas de una gran base de datos o analizando cientos de factores en un segundo.

En este tipo de juegos, y al igual que ocurre en una partida de ajedrez o damas, el jugador debe plantear la siguiente acción a realizar, mientras el contrario (ordenador o humano en un sistema multijugador) se limita a esperar los resultados del movimiento.

Este tipo de estrategia tiene infinidad de adeptos en todo el mundo y, hoy día, existen muchos títulos que, a pesar del enorme éxito del género en tiempo real, siguen ocupando un lugar importante en el mercado.

LOS PRIMEROS JUEGOS Y LA EVOLUCIÓN TÉCNICA

Realmente, debemos empezar por aquellos títulos que convirtieron los juegos de tablero al ordenador en la década de los 90. Entre ellos destacamos *Warhammer: Shadow of the Horned Rat* (Mindscape, 1996), fiel al juego de tablero más popular en el mundo de la estrategia. Antes de *Warhammer* aparecieron otros títulos ambientados en la segunda guerra mundial como *Steel Panthers* (MindScape, 1995) o la serie de campañas *Camping Series* de Talonsoft. No olvidemos que, al margen de la fantasía y las guerras mundiales, se encontraban

NOTA

▸ MEZCLANDO TIEMPO REAL Y TURNOS

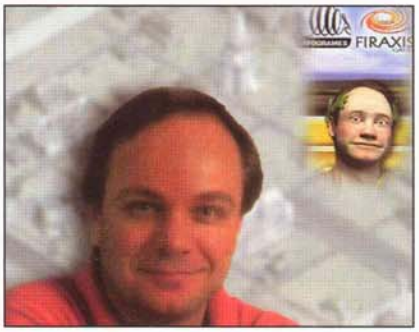
En ocasiones los juegos de estrategia por turnos se vieron obligados a luchar en el mercado con la creciente demanda del género en tiempo real. Aparecieron entonces títulos que daban la posibilidad al jugador de elegir entre una forma de juego y otra. Los títulos más representativos fueron *M.A.X. 2* (Interplay, 1998) y *X-COM: Apocalypse* (Microprose, 1997).

LA BIOGRAFÍA...

SID MEIER

Creador de **Civilization**

Sin duda, un maestro entre maestros. Enseñó su ordenada filosofía de diseño a otros maestros de la talla de Brian Reynolds *(Civilitation II)* o Bruce Shelley *(Age of Empires)*. A principio de los 80, cofundó la empresa Microprose, líder en el desarrollo de simuladores de vuelos de combate como: *Hellcat Ace* (1983), *F-15 Strike Eagle I* y *II, Silent Service*, etc. Es un gran aficionado a los juegos de estrategia de mesa, y lo reflejó en lo que sería la obra que sentó las bases de este género en ordenadores: *Civilization*. A partir de ahí, todo cambió. Siguieron más títulos como: *Colonization, Sid Meier´s Alpha Centaury, Raildroad Tycoon, Sid Meier´s Civilitation III*, etc. Abandonó Microprose en 1996 para fundar y dirigir la actual Firaxis con Brian Reynolds y Jeff Briggs. Con 48 años sigue al pie del cañón diseñando y dirigiendo juegos de indiscutible éxito mundial.



1

Warhammer y Steel Panther.

2

UFO de la serie X-COM y X-COM: Apocalypse.

3

Serie Worms.

4

Serie Civilization.

títulos ambientados en la Edad Media como la fantástica serie *WarLords* (Red Orb, 1990-1997). Microsoft y su afán de promover las DirectX y las nuevas posibilidades multimedia de los PCs publicó *Close Combat* (1996), que contribuyó a la aparición de gráficos más realistas y con mejores movimientos en los juegos de estrategia por turnos tradicionales, en los que aún se representaban las celdas de los tableros en pantalla.

En el año 1994, la casa Microprose rompió con el clásico manejo de ejércitos enteros, común en este género, publicando *Ufo* (1994) de su serie *X-COM*, en el que el jugador sólo maneja un grupo de soldados para luchar contra una invasión extraterrestre.

Todos estos títulos se caracterizaban por necesitar pocos recursos para su correcto funcionamiento; aún hoy es posible encontrar juegos de estrategia por turnos en donde los requisitos son mínimos. Sin embargo la llegada del 3D trajo aires de cambio para la representación de las batallas. Ejerció el liderato el juego *Panzer General 3D Assault* (SSI, 2000) y le siguieron otros como *Combat Misión* (Big Time Software, 2000) o *Battle Isle: The Andosia War* (Blue Byte, 2001). Aun así, la perspectiva isométrica y cenital seguían siendo la mejor elección para este tipo de juegos, aún utilizada en el 2000 por Microprose en *X-COM: Genesis*.

Ya en los ordenadores de 8 bits apareció un tipo de juegos de estrategia por turnos muy peculiar que se salía de los cánones establecidos en este tipo de género. Con una representación en 2D y generalmente con vista lateral, consistía en la lucha de un jugador o varios (en el mismo ordenador) con otro/s (humano u ordenador) en un escenario irregular definido aleatoriamente en cada nueva partida. Antes de empezar, cada jugador debía armarse según los créditos que poseía. Básicamente, en cada turno los jugadores estudiaban las estrategias de ataque o defensa según su arsenal. Para PC este tipo de juegos triunfó mundialmente con *Worms* (Team 17, 1995). *Worms* - trasladada a multitud de plataformas como PC, Amiga, PlayStation, Macintosh, etc.- vendió más de un millón de ejemplares y generó una serie de más de una docena de títulos *Worms*: *Worms 2*, *Worms Armageddon*, *Worms World Party*, etc. Con una cinemática fantástica y unos gráficos de dibujos animados, permitía jugar al mismo tiempo hasta 16 jugadores. Además, en la edición *Worms Word Party* es posible jugar a través de internet.

El desarrollo de juegos de estrategia por turnos corría también paralelo a temas que no eran de carácter bélico. Más bien, se acercaban de alguna manera a los simuladores de Dios, ya que el objetivo era prácticamente el mismo: crear y gestionar ciudades, países o civilizaciones enteras. Esta temática de juego en el género de la estrategia por turnos nació con *Civilization* (Micropose, 1992), bastión del género, desarrollado por Sid Meier, que sentó las bases de los juegos de estrategias actuales para ordenador. Con *Civilization* era posible controlar la evolución de una civilización desde cero a partir de un mapa vacío y herramientas para llenarlo. Cada jugador tenía en sus manos el poder de crear su propio mundo y, por turnos, cuidar de él. A *Civilization* siguieron otros títulos de Meier como *Colonization* (1995) y *Civilization II* (1996). Muchos acapararon los estantes siguiendo estos pasos como Sids Meier´s *Alpha Centauri* (Firaxis, 1999), *Imperium Galactica* (Digital Reality, 2000) o Civilization *III* con soporte para internet en *Civilization III: Play the World*.

En el próximo número...

... hablaremos de los juegos de estrategia en tiempo real (RTS).

Cuestionario Videojuegos 10

Preguntas

1. **Describe los pasos para crear el lado de un cuadrado en Blitz3D usando dos triángulos.**
2. **Hemos modelado un hombre, el cual contiene 6 partes: cabeza, tronco, 2 brazos y 2 piernas. ¿Cómo podemos cargarlo desde Blitz3D y texturizar sólo la cabeza?**
3. **¿Cómo podemos definir un FPS (rendimiento) estándar para un juego?**
4. **Para implementar un radar en 2D de un juego en 3D, ¿qué relación existe entre las coordenadas de ambos modos de representación?**
5. **Describe los pasos necesarios para crear un mapeado UV en LithUnWrap de un modelo.**
6. **¿Cómo podemos grabar en disco la textura pintada sobre un modelo en Deep Paint 3D?**
7. **Pasos básicos para realizar un loop en Rebirth.**
8. **Pasos básicos para realizar un loop en Fruity Loops.**
9. **¿Cómo podemos cambiar el volumen de una pista de audio en diferentes puntos en el multipista de Cool Edit Pro 2?**
10. **Una vez mezcladas todas las pistas en el multipista de Cool Edit Pro 2, ¿cómo podemos obtener un solo fichero de audio?**

Respuestas al cuestionario 9

▷ **1.** Escalando la entidad terreno como sigue:

```
ScaleEntity terreno,10,300,10
```

▷ **2.** Cargamos, por ejemplo, dos texturas:

```
   textura1=LoadTexture("textura_suelo1.png")
   textura2=LoadTexture("textura_suelo2.png")
Escalar la textura 1 y aplicarla al terreno:
   ScaleTexture textura1,512,512; Tamaño de la textura 512X512
   EntityTexture Terreno,textura1,0,0
Escalar la textura 2 y aplicarla al terreno:
   ScaleTexture textura1,128,128; Tileado (entre 2 hasta 512 siempre Múltiplo de 2)
   EntityTexture Terreno,textura2,0,1
```

▷ **3.**

```
Print "Raiz cuadrada de 25 es: "+Raiz_Cuadrada(25)
End
Function Raiz_Cuadrada(numero)
   Return Sqr (numero)
End Function
```

▷ **4.** En primer lugar hay que crear y situar una cámara para ver el entorno 3D

```
   Camara=CreateCamara()
   CameraRange Camara,0.1,Rango_Max_Vision
Seguidamente hay que iluminar el entorno:
   AmbientLight 150,150,150
   Sol=CreateLight()
Y para finalizar crear, texturizar y colocar el terreno:
   Terreno=LoadTerrain("Terreno1.png"); Mapa de alturas de 512x512
   ScaleEntity Terreno,10,300,10; Por ejemplo
   TerrainDetail Terreno,Num_polígonos,True
   Textura=LoadTexture("Textura_terreno1.png")
   ScaleTexture Textura, 512, 512
   EntityTexture Terreno,Textura
   PositionEntity Terreno,X#, Y#, Z#
```

▷ **5.** (A) Crear el esqueleto a partir de uniones.
(B) Asignar los vértices del modelo a las uniones.
(C) Mover las uniones y crear keyframes.

▷ **6.** Las animaciones se crean en el *Keyframer* en el modo *Animation* y deben ir entre *KeyFrames* a lo largo de la línea de tiempo.

▷ **7.** El ruido lo podemos solucionar eliminándolo o reduciéndolo aplicando el efecto *Noise Reduction*.

▷ **8.** En GoldWave la calidad de un sonido se puede cambiar realizando un resampleo con la función *Resample* en *Effects*.

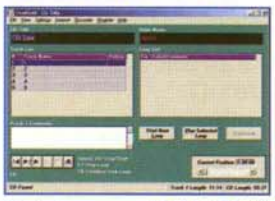
▷ **9.** Un cambio del volumen en Cool Edit Pro significa modificar la envolvente del sonido por medio de la creación de uno nuevo a través de la función *Create Envelope* situada en *Effects \ Amplitude \ Envelope*.

▷ **10.** (A) Crear un Script nuevo y darle un nombre al archivo en la ventana *Scripts and Batch Processing*.
(B) Añadir acciones al nuevo archivo por medio del botón "Record".
(C) Una vez grabadas las acciones, hay que registrar el nuevo script pulsando en el botón "<<Add to Collection>>".

Contenido CD-ROM 10

▶ AUDIO

■ Slow Gold 7.1



Cambia el tempo de tus composiciones sin que éstas pierdan su ritmo.

■ Advanced MP3/WMA Recorder 3.2.6

Grabación de audio en cualquier formato y en modo streaming.

■ Alcohol 1.2.3

Útil herramienta para realizar backups de CDS de audio, para que no pierdas nada de lo que compongas.

■ Blaze Media Pro 3.01

Edita el sonido y conviértelo al formato que desees.

■ Fruity Loops 3.3.0

Nueva oportunidad para conseguir la demo de este maravilloso programa.

■ Rebirth

También te ofrecemos de nuevo la ocasión de hacerte con esta excelente aplicación.

▶ DISEÑO 2D

■ Picasa 1.0.1

Organiza automáticamente todas tus imágenes con este potente clasificador.

■ Document Imaging Application 2.1

Sencilla aplicación para adquirir imágenes y archivarlas.

■ EyeBatch 2.0.12

Aplica múltiples y complejos efectos a tus imágenes.

■ Ashampoo Illuminator 1.50

Clasifica, visualiza y ten ordenadas todas tus imágenes fácilmente.

■ Fractal Fantasy 1.5

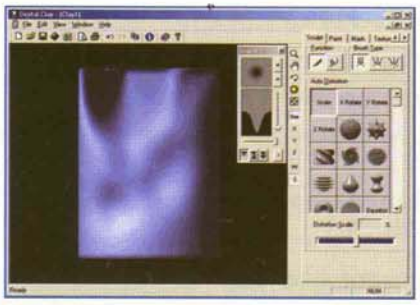
Crea fractales realistas o artísticos para dibujar árboles, nubes, paisajes, etc...

■ IMatch 3 3.1

Editor de imágenes muy profesional que te permitirá obtener imágenes con una calidad excepcional.

▶ DISEÑO 3D

■ Digital Clay 1.60



Diseña formas en tres dimensiones de un modo sencillo y muy intuitivo.

■ EBook 3D Wizard 4.6

Con esta aplicación podrás crear las imágenes en 3D para la portada del CD del juego.

■ Infinity Textures 2.3

Crea un montón de texturas para tus modelos en 3D.

■ 3D Studio

Demo del célebre programa de creación de modelos en 3D.

▶ PROGRAMACIÓN

■ Tarma Installer 2.0

Crea un instalador para tu programa fácil y rápidamente.

■ Soloway Controls 1.0

Diseña fácilmente las pantallas del juego gracias a este utilidad.

■ Visual WinHelp 1.0

Con esta aplicación podrás crear ventanas de ayuda de Windows, lo que te será muy útil para tu juego.

■ Tile Studio 2.33

Programa muy útil para crear tiles y sprites así como un editor de mapas.

■ Spriteworks 1.0

Original librería de gráficos para sprites que podrás usar en tus porgramas.

▶ JUEGOS

■ Civilization 2: Test of time

Segunda parte del mítico juego de la saga Civilization, bandera en los juegos de estrategias por turnos no bélicos.

■ Panzer General 3D

Uno de los primeros juegos de estrategia por turnos que usó las tres dimensiones.

■ Warhammer

El clásico juego estratégico de tablero, en su versión para PC.

■ Worms World Party

Excelente juego, secuela del mítico Worms, que tantos adeptos tuvo.

■ Zone of Fighters

Como todas las semanas, nuestro juego.

▶ VÍDEO

■ FX Movie Joiner 4.8.2

Con esta interesante utilidad podrás mezclar sin problemas archivos de vídeo con imágenes.

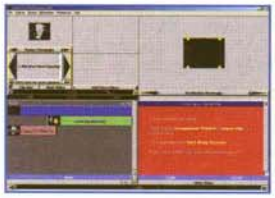
■ Honestech MPEG Editor 3.0

Robusto editor de vídeo que te será de gran ayuda.

■ Random Frame 1.2

Extrae los frames que desees de tus vídeos.

■ Zwei-Stein Video Editor 3.01.



Más de 64 efectos que podrás añadirle a tus vídeos para darles un toque muy personal.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.

Controlando al protagonista (I)

En esta primera entrega acerca de cómo implementar a nuestro protagonista en el juego, empezaremos por controlar su movimiento y el de la cámara, tratando también la actualización de su estado durante la partida.

MOVIENDO A NUESTRO PROTAGONISTA

Antes de empezar, vamos a situar todo el procedimiento de control del jugador en la función "control_jugador_principal()" en el módulo "jugador_principal.bb".

La bionave de combate es movida por el jugador mediante el ratón y/o de las teclas del cursor. Debemos, sin embargo, estudiar previamente cómo se ha diseñado este movimiento para saber qué instrucciones debemos utilizar. Sabemos que la premisa más importante es que la bionave siempre está en movimiento. Por lo tanto, lo más sensato será utilizar el comando "MoveEntity" para desplazarla. Otra cuestión es cómo la movemos. Básicamente, todo el control de la bionave se centra en el uso del ratón a través de su desplazamiento y la pulsación de los botones, aunque podemos añadir el control por teclado fácilmente. Pero centrémonos en el uso del ratón. A pesar de que la obtención de datos a través de periféricos externos la estudiaremos más detenidamente en la sección "Blitz3D" de una próxima entrega, adelantaremos su uso en este número de una forma sencilla y fácil de comprender.

ROTANDO LA BIONAVE CON EL RATÓN

Lo que primero nos interesa es cómo obtener el desplazamiento del ratón y convertirlo en datos numéricos, los cuales servirán para modificar las coordenadas de la bionave. Solamente nos interesa el movimiento de los ejes horizontales del ratón para modificar la rotación de la nave en su eje Y. Así que, desplazándolo de izquierda a derecha, hacemos que la bionave gire para cambiar de rumbo ya que, si recordamos, siempre está en movimiento.

Es importante resaltar que, para crear un punto de rotación central en la nave, es imprescindible crear un pivote. El procedimiento para su creación lo implementamos en el momento de cargar el fichero y crear el modelo en la función "crear_modelos()" del módulo "funcpantaudio.bb":

```
Function crear_modelos()
  bionave=LoadMesh ("c:\zone of
  fighters\modelos\bionave.3ds")
  pivote_bionave=
  CreatePivot(bionave)
...
```

De vuelta al módulo de control, comenzamos la función almacenando en la variable flotante YY# el desplazamiento horizontal absoluto del ratón mediante la función "MouseXSpeed", la cual detecta los cambios en su movimiento. El valor obtenido por esta función es un número entero; debemos convertirlo en un número flotante y dividirlo entre 7 para obtener valores decimales y así tener incrementos o decrementos de desplazamientos razonables.

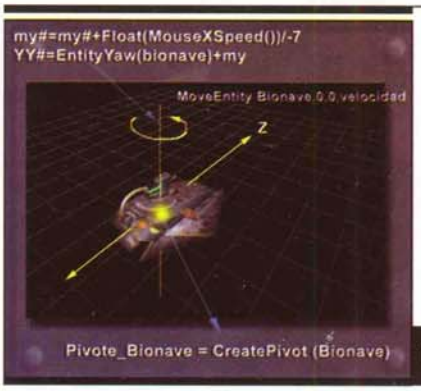
```
my#=my#+Float(MouseXSpeed())/-7
YY#=EntityYaw(bionave)+my
```

A continuación, ejecutamos la rotación de la bionave y su sombra en su eje Y. Además, hemos tenido en cuenta un incremento extra en la velocidad de rotación cuando la cámara se encuentra en el modo subjetivo o primera persona, controlado por la variable global "tipo_camara":

```
RotateEntity bionave,0,yy+
MouseXSpeed() , 0
```

De esta forma, todo el proceso queda como sigue:

```
If tipo_camara=1
    my#=my#+Float(MouseXSpeed())/-7
    YY#=EntityYaw(bionave)+my
    RotateEntity bionave,0,yy+
    MouseXSpeed(),0
    RotateEntity sombra_bionave,
    0,yy,0
  Else
    my#=my#+Float(MouseXSpeed())/
    -7 : my=my/2 ;➡ Mitad de
    velocidad de giro para 3ª
    persona
    YY#=EntityYaw(bionave)+my
    RotateEntity bionave,0,yy,0
    RotateEntity sombra_bionave,
    0,yy,0
EndIf
MoveMouse GraphicsWidth()
Shr 1,GraphicsHeight() Shr 1
```



Representación gráfica de la rutina de rotación y desplazamiento de la bionave.

La función "MoveMouse" colocará el cursor (invisible) siempre en el centro de la pantalla, así nunca perderemos el desplazamiento del ratón (Fig. 1).

DESPLAZAMIENTO Y ACELERACIÓN

En el diseño del juego contemplamos la idea de que la bionave del protagonista se encuentra en continuo desplazamiento. El único control que el jugador puede tener sobre este desplazamiento es aumentar o disminuir la aceleración. Para ello, elegimos de nuevo el ratón como vehículo principal de control. Utilizaremos entonces el botón derecho para aumentar la aceleración. Si lo pulsamos aumentará y si lo soltamos disminuirá hasta una velocidad inicial establecida. También debemos contemplar la posibilidad del uso del teclado por parte del jugador, ya que siempre es necesario implementar en el juego todas las opciones posibles de control. Así que utilizaremos la tecla del cursor "arriba" para aumentar también la aceleración.

Las variables que utilizaremos para dotar a la bionave de movimiento son: *velocidad#* y *aceleración#*. Ambas son inicializadas en "Definiciones.bb" con los valores 0.8 y 0.2 respectivamente.

Para implementar este control utilizamos una sentencia condicional:

```
If (MouseDown(boton2) Or
KeyDown(200)) Then
    velocidad#=velocidad+aceleracion#
    MoveEntity bionave,0,0,velocidad
  Else
    velocidad=velocidad-(aceleracion#*2)
    MoveEntity bionave,0,0,velocidad
EndIf
```

Si pulsamos el botón derecho del ratón o pulsamos el cursor "arriba" incrementamos la velocidad según el valor contenido en *aceleración#* (0.2). Si no, reducimos la velocidad el doble de rápido que aceleramos:

```
velocidad=velocidad-(aceleracion#*2)
```

De esta forma, evitamos una desaceleración brusca y conseguimos un movimiento suave. Pulsemos el botón o no, siempre desplazamos la bionave en su eje Z; es decir, hacia delante o hacia atrás, ya que el resto de direcciones lo determinará el giro:

```
MoveEntity bionave,0,0,velocidad
```

Es necesario añadir una condición más a la anterior sentencia. Se trata de controlar la velocidad máxima para que al acelerar haya un tope. Utilizaremos otra variable llamada *velocidad_max%* la cual inicializamos con un valor de 15:

```
If (MouseDown(boton2) Or
KeyDown(200)) And velocidad <
velocidad_max Then
```

De esta manera conseguimos que la velocidad actual de la bionave no supere el valor 15.

A su vez, hemos de controlar también la velocidad mínima, por ejemplo 4; de esta manera garantizamos un desplazamiento mínimo continuo. Debemos añadir, seguidamente, una sentencia más:

```
If velocidad<4 velocidad=4
```

Antes de controlar el factor de gravedad, debemos actualizar las coordenadas de la bionave y la posición de la cámara llamando a la función "mueve_camara":

```
mueve_camara (bionave,
velocidad_seguimiento)
```

Para el correcto funcionamiento de la función es imprescindible pasarle como parámetros la entidad a la cual seguirá, en este caso la bionave, y a la velocidad que lo hará ("velocidad_seguimiento") (Fig. 2).

CONTROLANDO LA CÁMARA. FUNCIÓN "MUEVE_CAMARA"

El juego soporta cuatro vistas de cámara diferentes: subjetiva, tercera persona, cenital y seguimiento en tercera persona. En todos los casos existe una particularidad común y es que la cámara sigue en todo momento a la bionave. Para realizar este procedimiento utilizaremos la instrucción "PointEntity", la cual implementaremos en la función "mueve_camara" situada en el módulo "Fjuego.bb".

La distancia entre la cámara y la bionave en 3ª persona depende de la variable "Velocidad_seguimiento".

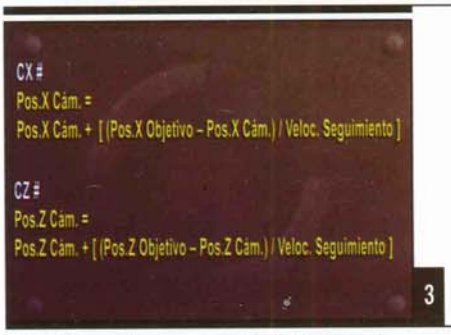
Los parámetros de entrada de esta función son: el objetivo a seguir ("objetivo") y la velocidad de seguimiento de la cámara ("velocidad_camara"):

```
Function mueve_camara
(objetivo, velocidad_camara)
```

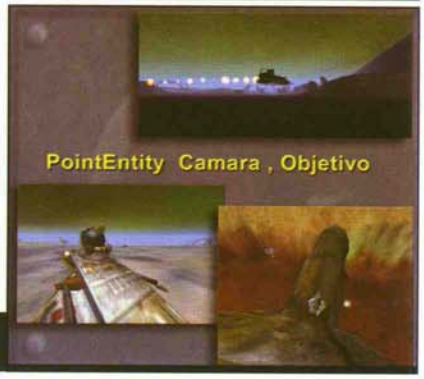
Posteriormente actualizamos la posición de la bionave almacenándola en las variables *X#* , *Y#* y *Z#*:

```
x#=EntityX(bionave)
y#=EntityY(bionave)
z#=EntityZ(bionave)
```

Continuamos, calculando las coordenadas de la cámara con



Fórmulas empleadas para calcular las coordenadas de la cámara con respecto al objetivo (bionave). Además, permiten desplazar y rotar la cámara de una forma suave.



4

Independientemente de la vista escogida, la cámara siempre apuntará al objetivo (bionave) mediante la instrucción "PointEntity".

respecto al objetivo a seguir. Para ello, determinaremos un desplazamiento según la posición actual de la cámara y del objetivo y lo almacenaremos en las variables *CX#* y *CZ#*:

```
cx#=EntityX(camara)
cz#=EntityZ(camara)
objetivo_x#=EntityX(objetivo)
objetivo_z#=EntityZ(objetivo)
```

objetivo_x# y *objetivo_z#* almacenarán la posición del objetivo a seguir, en este caso la bionave, que servirán luego para actualizar la posición de la cámara de una forma suave y siempre dependiendo del contenido de la variable *velocidad_camara*. Para implementar estos desplazamientos utilizaremos las fórmulas que se muestran en la figura 3.

```
cx#=cx#+((objetivo_x#-cx#)
/velocidad_camara)
cz#=cz#+((objetivo_z#-cz#)
/velocidad_camara)
```

La altura de la cámara dependerá del tipo de vista que se haya seleccionado. De tal modo que, si la vista es subjetiva -en primera persona-, debemos situarla al nivel de la cabeza del protagonista. Para ello, situaremos la cámara 4 puntos más arriba del centro vertical de la bionave; es decir, "Y# + 4". Para el resto de las vistas, precisaremos en todo momento de la altura del terreno. Además, es necesario añadirle una altura extra para evitar una visión a ras de suelo y de esta manera situar la cámara por encima de la bionave. Debemos entonces utilizar una variable para almacenar esta altura adicional: *distanciaY_camara%*.

Este valor será el que determinará los tipos de vista cenital, tercera persona o seguimiento. Por defecto asignamos un valor de 158 en el módulo de "Definiciones.bb". Almacenaremos este valor en la variable *CY#* que será la que utilicemos finalmente:

```
cy#=distanciaY_camara
If tipo_camara=1 ; Primera persona
  PositionEntity camara,cx,y+4,cz
Else ; Otras vistas
  PositionEntity Camara,cx,
  TerrainY(terreno1,EntityX
  (bionave),EntityY(bionave),
  EntityZ(bionave))+cy,cz
EndIf
```

Finalmente, nos queda apuntar la cámara hacia el objetivo y actualizar la variable *y_bionave#*, la cual nos ayudará a situar la sombra de la bionave, ya que almacena la posición de ésta sobre el terreno:

```
PointEntity camara,objetivo
y_bionave=TerrainY(terreno1,
x,y+20,z)
```

CAMBIANDO LAS VISTAS DE CÁMARA. FUNCIÓN "CAMBIAR_CAMARA"

Dentro del módulo "Fjuego.bb", hemos situado, a continuación de la anterior función, un procedimiento para actualizar la situación de la cámara en las distintas vistas disponibles para el usuario. La función adquiere el parámetro del tipo de cámara elegido almacenado en la variable global *tipo_camara*:

```
Function cambiar_camara (tipo_camara)
```

Después, simplemente utilizamos una sentencia "Select .. Case" para modificar los parámetros de la situación de la cámara según el tipo elegido.

Hay otros factores a tener en cuenta a la hora de cambiar las vistas y son controlar la visualización de la bionave, su sombra y cambiar el zoom de la cámara. Por lo tanto, en la vista subjetiva debemos ocultar la entidad bionave y su sombra para poder dar la sensación deseada. Para ocultar la bionave cambiaremos su opacidad a 0 con "EntityAlpha bionave, 0 " y para hacer lo mismo con la sombra utilizaremos " HideEntity sombra_bionave ". Es necesario controlar también si el jugador tiene activado o no el camuflaje, examinando la variable *camuflaje_bionave*:

```
Select tipo_camara
  Case 1
    distanciaZ_camara=-2
    velocidad_seguimiento=6
    EntityAlpha bionave,0
    HideEntity sombra_bionave
    CameraZoom camara,.54
  Case 0
    distanciaY_camara=107
    distanciaZ_camara=0
    velocidad_seguimiento=15
    If camuflaje_bionave=0
      EntityAlpha bionave,1
    Else
      EntityAlpha bionave,.1
    EndIf
      ShowEntity sombra_bionave
      CameraZoom camara,.8
  Case 2
    distanciaY_camara=700
    distanciaZ_camara=0
    velocidad_seguimiento=63
    If camuflaje_bionave=0
      EntityAlpha bionave,1
    Else
      EntityAlpha bionave,.1
```

5

Dos ejemplos de la vista en tercera persona.

6

Algunos ejemplos de vista de seguimiento en tercera persona.

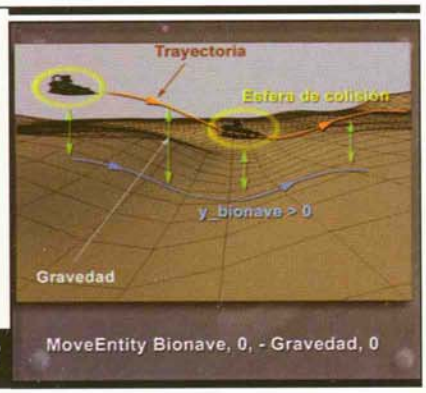
```
    EndIf
      ShowEntity sombra_bionave
      CameraZoom camara,.8
    Case 3
      distanciaY_camara=62
      distanciaZ_camara=-139
      velocidad_seguimiento=12
    If camuflaje_bionave=0
      EntityAlpha bionave,1
    Else
      EntityAlpha bionave,.1
    EndIf
      ShowEntity sombra_bionave
      CameraZoom camara,.54
End Select
```

Para terminar la función es preciso devolver el valor de la variable *tipo_camara*:

```
Return (tipo_camara)
```

CONTROLANDO EL FACTOR GRAVEDAD

Para determinar un entorno creíble es importante proporcionar ciertos elementos físicos, como por ejemplo la gravedad. Implementarla es realmente fácil. Sólo tenemos que mover a la bionave continuamente hacia el suelo decrementando su coordenada Y con el valor de "gravedad", pero sólo si está sobre el suelo; es decir, si supera el valor "Y=0":



7

Esquema gráfico del funcionamiento del control de la gravedad.

```
If y_bionave>0
  MoveEntity bionave,0,-gravedad,0
EndIf
```

De todas formas, el sistema de colisiones tiene prioridad frente al sistema de coordenadas. Por lo tanto, aunque siguiéramos decrementando la coordenada Y, la bionave se pararía al tocar con el terreno. La sentencia condicional anterior sólo nos sirve para evitar, precisamente, que la coordenada "y_bionave" baje del valor 0 (Fig. 7).

Explicaremos la implementación de las colisiones con el entorno en una próxima entrega.

AÑADIENDO DETALLES. SOMBRA Y COMBUSTIÓN

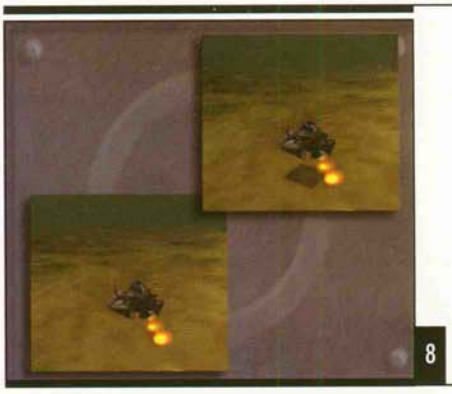
Para terminar, podemos añadir otros detalles, como la sombra de la bionave o la combustión del motor.

El hecho de añadir una sombra a la nave no se debe sólo a cuestiones estéticas, ya que la bionave se desplaza sobre el terreno en suspensión. Por lo tanto, es fundamental implementarla para proporcionar información adicional al jugador de la altura de la bionave. Utilizando, pues, la variable *y_bionave*, obtenemos la altura actual de ésta sobre el terreno. De este modo, sólo tenemos que colocar la sombra según los ejes X, Y y Z de la bionave y 15 puntos más abajo con referencia al eje Y (Fig. 8).

Si "y_bionave" vale "TerrainY(terreno1,x,y+20,z)", debemos posicionar la sombra en "y_bionave+5":

```
PositionEntity sombra_bionave,
x,y_bionave+5,z
```

Otro factor a tener en cuenta es proporcionar el efecto de la combustión del motor. Este hecho reforzaría aún más la idea de que la bionave está continuamente en movimiento. La mejor manera de implementar la combustión es a través de un emisor de partículas situada en el tubo de escape. Pero de este elemento hablaremos con mucho más detalle en una próxima entrega (Fig. 9).



8

A diferencia de la figura inferior, en la superior se puede apreciar la distancia de la bionave al suelo gracias al uso de sombras.

Hay muchos más aspectos que pueden ser implementados en las rutinas de movimiento y cámara del protagonista como pueden ser la aplicación de algún tipo de animación o colocar una nueva cámara que muestre la parte trasera de la bionave.

9

Un ejemplo de emisión de partículas para simular la combustión del motor.

En el próximo número...

... continuaremos controlando a nuestra bionave, programando las acciones de disparo, camuflaje y activación del escudo.

Fabricando los elementos del juego (II)

En este número terminaremos de realizar el Luny. Continuaremos fabricando el resto de seres que habitan los terrenos de combate como los "Shaark" y las plantas carnívoras "Dreecks".

CREANDO EL MAPEADO UV Y TEXTURIZANDO EL "LUNY" CON DEEP PAINT 3D

Para crear el mapeado UV realizaremos la misma operación que con el "Slunk" en LithUnWrap. Optimizaremos y aplicaremos un "UV mapping" "Planar" con la opción "Y top". Una vez salvado, lo cargamos con Deep Paint 3D para texturizarlo. Generamos un material nuevo con 256 x 256 de tamaño y aplicamos una imagen en blanco "Nothing" al canal de color ("C"). Ajustamos el modelo en la ventana y aplicamos un relleno con la textura *Skin 2* en *Texture Paints*. Para finalizar, pintaremos los detalles de la sangre y suciedad en la base de la planta utilizando la misma variación de pincel que en el modelo anterior; es decir, *Paste Oil +* en *Variations* con un pincel escalado al 15% (Fig.1).

Una vez pintado, exportamos la textura con el nombre "textura_luny.bmp" en el directorio "C.\juego_ZOF\luny\texturizado".

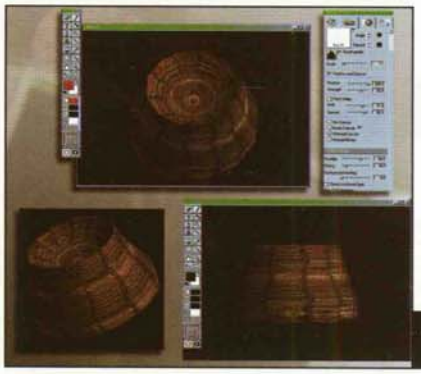
MODELADO DE UNA PLANTA SHAARK

Esta planta tiene un aspecto parecido al de las mazas utilizadas como armas en la Edad Media. No tiene movimiento alguno y aparecerá en el juego clavada en la tierra. Así que empezamos modelando el tronco que no es más que un cilindro de 2 "Stack" (Divisiones) y 8 "Slices". Para crear la base de la esfera de la planta, desplazamos los vértices que dividen las dos secciones del tronco hacia arriba. Seleccionamos los vértices superiores y, con la herramienta de escalado, los abrimos. A continuación, de los 8 vértices bajamos 4 de dos en dos y salteados como se muestra en la figura 2 para lograr una forma estrellada.

Para crear la cabeza de la planta elegimos una geoesfera de densidad 1 y la situamos sobre la base en forma de estrella que acabamos de modelar. Para formar los pinchos que irán clavados en la cabeza de la planta partimos de cilindros de una sola división y de 3 "Slices". Seleccionamos los vértices superiores y los unimos con "Ctrl" + "N" para formar la punta. Una vez modelada la espina la colocamos en cualquier lugar de la cabeza. Ya sólo nos queda duplicarla varias veces y repartir las nuevas espinas por la superficie de la esfera realizando operaciones de rotación y desplazamiento. Finalizamos, seleccio-

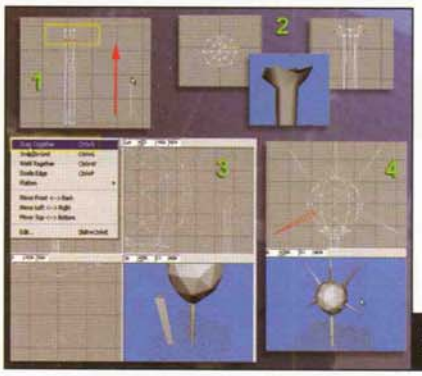
TRUCO

Podemos abrir una ventana con la perspectiva del modelo en LitUnWrap con la opción *Show Model* en el menú *Preview*.



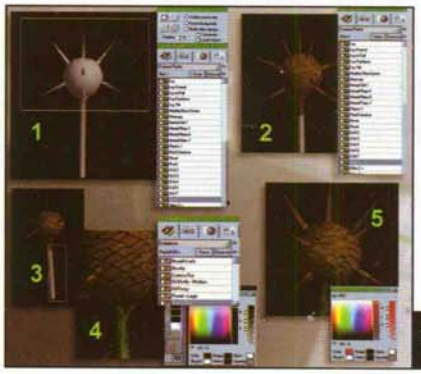
1

Después de cubrir con una textura el modelo es interesante la aplicación de detalle por medio del pincel.



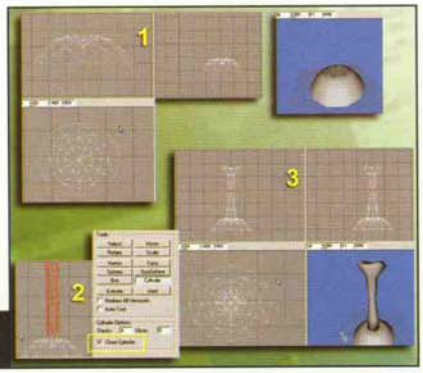
2

Procedimientos para el modelado de una planta Shaark.

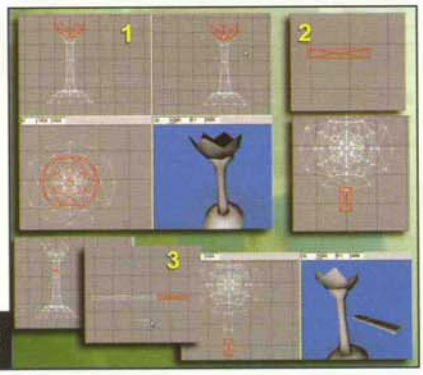


3

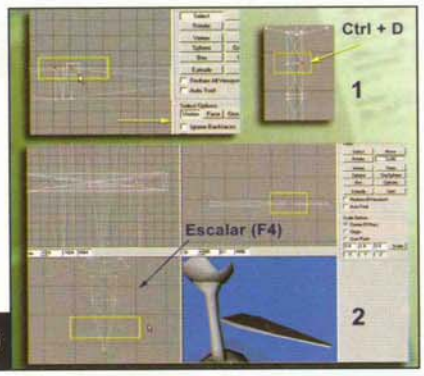
Procedimientos para el texturizado de un Shaark.



4 *Los anillos de un cilindro nos servirán para dar forma al tallo de la planta.*



5 *La mitad de una geoesfera es ideal para obtener la base de la boca.*



6 *Para obtener la forma de las hojas a partir de tres cubos unidos podemos aplicar las herramientas de escalado y rotación en las uniones.*

nando todas las piezas y agrupándolas para exportar como un sólo *mesh* y en formato .ms3D con el nombre "shaark.ms3d" en el directorio "C:\juego_ZOF\shaark\modelado\"

CREANDO EL MAPEADO UV TEXTURIZANDO CON DEEP PAINT 3D

El mapeado UV lo crearemos de nuevo con LithUnWrap, pero en esta ocasión, y una vez optimizado, aplicaremos un "UV Mapping" "Planar" en su eje X, para conseguir una plantilla que muestre una disección del modelo.

Exportamos en formato .OBJ y posteriormente lo cargamos en Deep Paint 3D para texturizarlo.

Al igual que en el modelo anterior, generamos una nueva textura de dimensiones 256 x 256 y aplicamos "Nothing" en el canal "C". Vamos a pintar con aspecto diferente el tronco y la cabeza de la planta. Para empezar, ajustamos el modelo a la vista y seleccionamos toda la cabeza y los pinchos con la herramienta de selección rectangular *Rectangle tools*. Elegimos la textura *Skin 1* en *Texture Paint* de *Command Panel* y aplicamos la herramienta de relleno. Seguimos con el tronco. Una vez seleccionado le aplicamos un relleno con la textura *Skin 3+*, la cual hemos modificado previamente el tono ("Hue") a 61 y el tamaño horizontal ("Scale X") a 13.25. Como siempre, aplicaremos algunos detalles para darle a la planta un aspecto más real. Empezamos por perfilar la base de la planta con el pincel y eligiendo un color verdoso y el efecto de pintado *Pastel Oil* + en *Variations*. Luego pintaremos manchas de sangre en las espinas y cuerpo y algo de suciedad en el tronco con el pincel al 15 % de su tamaño y con la característica *Pastel Oil*. No olvidemos elegir el color rojo para la sangre y un tono marrón para las manchas de suciedad (Fig. 3).

Una vez hayamos terminado, exportamos la textura contenida en el canal de color con el nombre "textura_shaark.bmp" en el directorio "C:\juego_ZOF\texturizado\shaark".

MODELANDO LA PLANTA CARNÍVORA DREECKS CON MILKSHAPE 3D

Para modelar la planta tenemos que definir una postura: optamos por hacerla totalmente rígida, mirando hacia arriba y con los tentáculos de los ojos extendidos. La sensación de movimiento se construirá posteriormente con el esqueleto en Character FX.

Empezamos por la base de la planta, la cual parte de una esfera con 6 "Stacks" y 12 "Slices". Seleccionamos la mitad de la misma sin elegir *Ignore Backfaces* y cortamos. Seguidamente, el vértice superior de la mitad de la esfera lo desplazamos hacia abajo para crear el hueco donde irá el tallo (Fig. 4 – 1).

Para crear el tallo partimos de un cilindro de 4 divisiones de altura (stack) y 8 de circunferencia (slices) Estas divisiones o anillos nos servirán posteriormente para dar alguna forma al tallo (Fig. 4 - 2). Ensanchemos, pues, la base del tallo seleccionando (F1) los vértices más inferiores con la herramienta de escalado (F4) en la vista superior. Hacemos la misma operación con los vértices de las divisiones impares siguientes desde abajo hacia arriba. Para la última división superior realizamos un escalado mayor que en las demás hacia fuera, ya que servirá como apoyo a la siguiente pieza que es la base de la boca (Fig. 4 - 3). Para esta pieza, utilizamos una geoesfera de densidad 1 y le

borramos la mitad superior de los vértices. A continuación, ajustamos la esfera a la parte superior del tallo, la cual habíamos ensanchado. El ajuste lo realizamos mediante operaciones de escalado y desplazamiento (Fig. 5 - 1).

Para modelar las hojas vamos a utilizar tres cubos aplastados, alargados y unidos entre sí. Sólo es necesario modelar uno de ellos con la herramienta de escalado. Los demás los iremos duplicando ("Ctrl" + "D") y colocando uno a continuación del otro para poderlos unir (Fig. 5 – 2,3). La unión la realizamos seleccionando el primer vértice superior de cada cubo ignorando los de atrás (*Ignore Backfaces* deseleccionado) y pulsando "Ctrl" + "D" (Snap Together). Hacemos lo mismo con los vértices inferiores. A continuación, elegimos los vértices traseros con *Ignore BackFaces* seleccionado y los pegamos también. El resto de los cubos los unimos de la misma manera (Fig. 6 – 1).

Para dar la forma a las hojas a partir de los tres cubos unidos, utilizaremos las herramientas de escalado y desplazamiento en los vértices que hemos unido. Por ejemplo, para obtener la punta de la hoja, seleccionaremos los vértices más exteriores en la vista superior y los escalamos hacia dentro (Fig. 6 – 2). Para obtener las formas curvas, elegiremos los vértices del centro, también en la vista superior, y los desplazaremos hacia un lado. Estos procedimientos se muestran gráficamente en la figura 7.

Para la otra hoja sólo tenemos que duplicar la que tenemos, seleccionándola completamente y pulsando "Ctrl" + "D". Seguidamente, hacemos un *Mirror Front <-> Back* en el menú *Vertex* Para evitar que sea igual que la primera, la modificaremos un poco mediante el desplazamiento de sus vértices y la colocaremos un poco más abajo en el tallo.

La boca del Dreeck está compuesta por dos grandes mandíbulas. Para modelarla, necesitaremos una esfera de 6 "Stacks" y 12 "Slices" dividida en dos. Para ello, tenemos que seleccionar la mitad de los vértices con *Ignore Backfaces* deseleccionado y borrarlos. Nos quedamos entonces con la mitad de la esfera que corresponderá a una de las mandíbulas. Sólo es necesario trabajar con una de ellas, ya que la otra la podemos obtener mediante un duplicado (Fig. 8 – 1,2).

Hay que tener en cuenta que, al cortar una esfera, la parte interior queda vacía; es decir, no hay ninguna cara poligonal y necesitamos cerrarla. Si cambiamos la vista de perspectiva en modo "Flat Shade" podemos observar este hecho (Fig. 8 – 3). Para cubrir el hueco dejado, necesitamos fabricar manualmente los polígonos que faltan. Partiremos, para realizar esta operación, de los vértices que forman el contorno interior de la semiesfera. Los seleccionamos y los desplazamos hacia un lado para poder unirlos. Para ello, elegimos la herramienta *Face* del grupo *Model* y unimos cada vértice entre sí de tres en tres para formar los polígonos que cerrarán la semiesfera (Fig. 9).

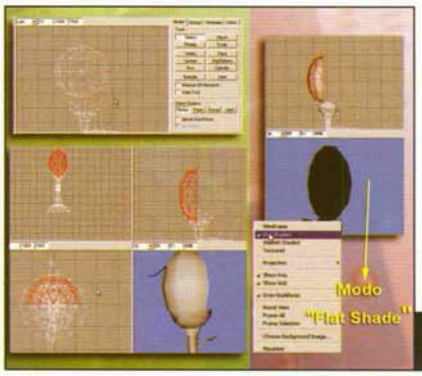
Una vez completada la operación, volvemos a colocar la cara interior de la semiesfera en su sitio seleccionando de

NOTA

Hay una opción en Milkshape 3D que nos permite saber la cantidad de polígonos que tiene nuestro modelo. Se encuentra en el menú *Tools* y la opción es *Show model Stadistics* (023).

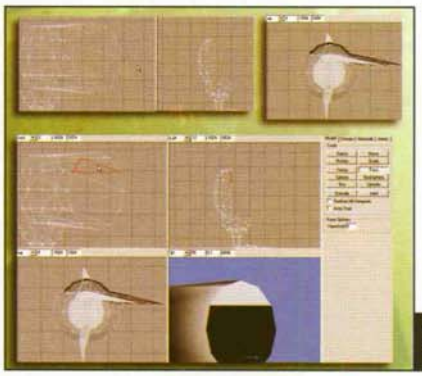
7

Mediante el desplazamiento individual de vértices podemos dar variadas formas a las primitivas.



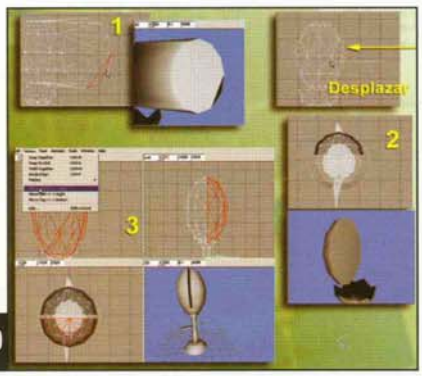
8

La mitad de una esfera puede servir perfectamente como mandíbula.



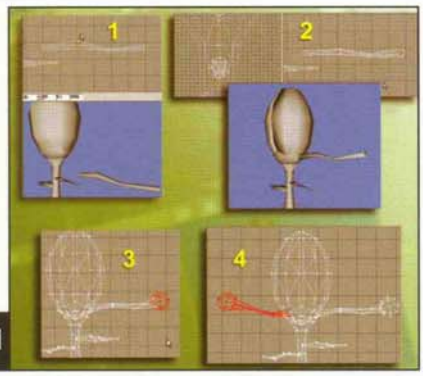
9

Una vez cortada la esfera es necesario cubrir de polígonos el hueco interior para obtener adecuadamente una de las mandíbulas.



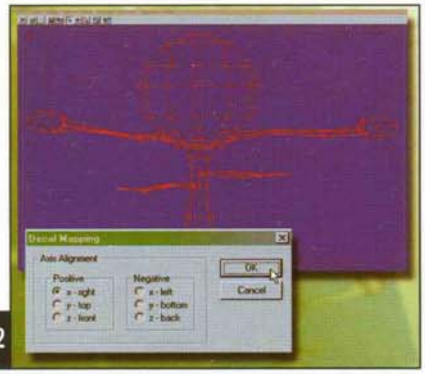
10

Una vez cerrada la semiesfera, la duplicamos para crear la otra mandíbula.



11

A partir de un cilindro y por medio de los procedimientos utilizados para crear las hojas es posible obtener los tentáculos oculares.



12

Antes de texturizar la planta es necesario crear el mapeado UV, en este caso con LithUnwrap.

nuevo todos los vértices y desplazándolos a su lugar original (Fig. 10).

El siguiente paso será ajustar la mandíbula a la base de la boca con las herramientas de escalado y desplazamiento. Una vez ajustada, duplicamos y hacemos un espejo horizontal (*Mirror Front<->Back*) para generar y colocar la otra mandíbula (Fig. 10 –3).

Las últimas partes que nos quedan son los tentáculos que albergarán a los globos oculares. Partimos de un cilindro con 3 partes que colocaremos horizontalmente en la vista "Left" rotándolo 90 grados. Para dar forma al cilindro y obtener un aspecto más orgánico, seleccionamos los vértices de cada parte del cilindro y los desplazamos al igual que hicimos con el modelado de las hojas. Debemos también preparar el extremo del tentáculo para albergar el globo ocular. Así que, mediante el desplazamiento de los tres vértices abrimos el extremo del cilindro como se muestra en la figura 11.

Para el otro tentáculo, realizamos de nuevo un duplicado y una simetría horizontal y ajustamos la nueva posición.

Para los globos oculares utilizamos una esfera de 4 "Stacks" y 4 "Slices". Para terminar: le asignamos un tamaño adecuado, la colocamos en el extremo del tentáculo, duplicamos y aplicamos de nuevo una simetría horizontal.

RECORDATORIO

Si al unir tres vértices para formar un polígono éste no se ve en la vista de perspectiva es que el orden de dichos vértices está al revés. Para solucionarlo, basta con elegir la opción "Face / Reverse Vertex Order".

Finalmente, debemos agrupar todas las partes, seleccionando el modelo completo y pulsando el botón "Regroup" en la herramienta de agrupamiento *Groups* (Fig.11 - 3,4).

Ya sólo nos queda salvar el modelo terminado en formato .OBJ para poder texturizarlo con Deep Paint 3D.

CREANDO EL MAPEADO UV

Antes de crear la textura de la planta es preciso general su mapeado UV. Aunque es posible hacerlo con Deep Paint 3D a través de la herramienta "MercatorUV", utilizaremos LithUnwrap, ya que nos permitirá también aplicar un optimizado de vértices.

Como es habitual, dentro de LithUnwrap, cargamos el modelo y elegimos la opción *Optimize Model* en el menú *Tools*. Una vez realizada la optimización, seleccionamos el objeto por completo con *Select / All* y de nuevo en el menú *Tools* elegimos *UV Mapping / Decal* y pinchamos la opción *Positive / x- right* para alinear los ejes hacia un lado.

Finalmente, exportamos el modelo en formato .OBJ y ya estamos preparados para texturizar nuestra planta carnívora (Fig. 12).

RECORDATORIO

UV Mapping o "mapeado de coordenadas UV" es una serie de posiciones colocadas en una imagen que se unen a puntos en un objeto 3D para ubicar una textura sobre él.

En el próximo número...

... realizaremos el texturizado y la animación del Dreeck.

Herramientas para hacer la música de un juego (II)

En el número anterior comenzamos a desarrollar un loop en Fruity Loops. Estudiamos lo fácil que es crear y copiar patrones, los cuales utilizaremos para componer la canción que servirá de loop. En esta entrega terminaremos nuestro loop y seguiremos conociendo nuevas características que hacen de esta herramienta una de las más sofisticadas e interesantes.

UTILIZANDO "PLAYLIST"

En Fruity Loops, la canción se escribe en la sección "Playlist". En ella, se crea una serie de listas con el orden de reproducción de cada patrón a lo largo de una línea de tiempo. Para visualizar "PlayList" elegimos la opción "View / Playlist" o bien "Ctrl" + "L". Es preciso activar el modo "Loop" para poder trabajar en "Playlist". Una vez activado, la reproducción ("Play") sólo afectará a "Playlist" y no a las pistas de patrones. "Playlist" está dividido por una cuadrícula. Cada cuadro corresponde a un compás. Horizontalmente lo forman los diferentes compases a lo largo del tiempo y verticalmente los diferentes patrones. De este modo, para construir la canción sólo tenemos que rellenar estos cuadros dibujando con el ratón. Podemos ver en la figura 1 la composición en "Playlist" para nuestro loop (este loop esta contenido en el CD con el nombre "loop_fruity.wav") (Fig. 1).

Una vez terminado el loop, sólo queda exportarlo en formato .WAV (aunque también es posible hacerlo en .MP3) Elegimos la opción *File / Export / Wave File ...* o pulsamos "Ctrl" + "R". Aparecerá una ventana flotante, donde podremos elegir la calidad para nuestro loop.

EDITANDO LAS PISTAS

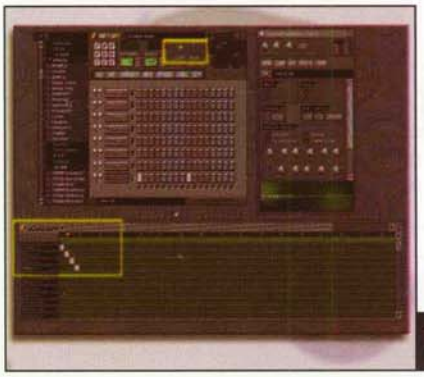
La posibilidad que Fruity Loop admite para la edición de las pistas es sorprendente y con un diseño eficaz. Antes de continuar, apuntar que en Fruity Loops las pistas de sonido se denominan *canales*. Disponemos de un editor gráfico que nos permite modificar los parámetros del volumen, panorámico, tono, frecuencia de corte, etc.. de cada canal. Con sólo pulsar sobre el botón aparecerá debajo del canal seleccionado un panel donde, mediante un lápiz, podemos dibujar los nuevos parámetros pulsando el botón izquierdo del ratón. Para borrarlos, utilizaremos el botón derecho. En la parte inferior del panel se encuentra un deslizador, que sirve para elegir los diferentes parámetros.

Para seleccionar otro canal, basta con pulsar en el "led" o lucecita verde que hay en el comienzo de los compases y para mutarla o desactivarla en la luz que se encuentra debajo de los botones de volumen y panorámico.

También disponemos de un editor gráfico de notas, el cual aparecerá debajo del canal seleccionado al pulsar el botón . Este panel tiene forma de pequeños teclados en el que podemos seleccionar el tono de nuestra nota (Ver Fig. 2).

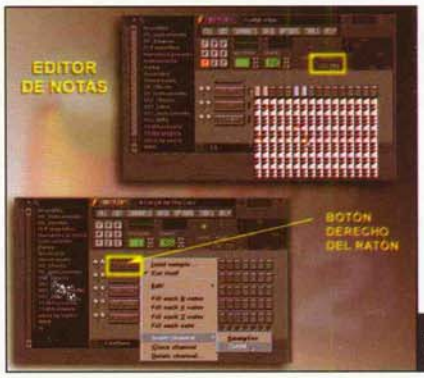
TIPOS DE CANALES

Fruity Loops trabaja con dos tipos diferentes de canales según su contenido: canales de muestras ("sampler") y canales de sonidos sintetizados ("TS404") El



1

En la sección "Playlist" es donde escribimos la canción que servirá de loop.



2

Editor gráfico de notas y procedimiento para insertar un nuevo canal.

NOTA

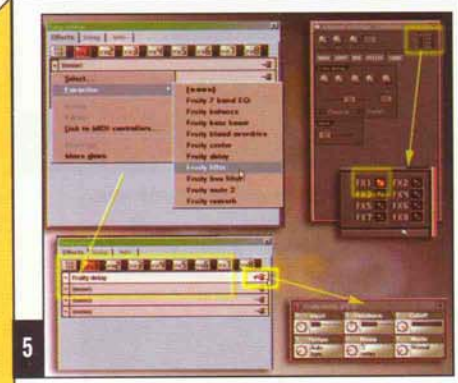
Es posible cargar varias muestras a la vez, seleccionándolas en el directorio con la tecla "Shift" pulsada. Al cargarse, los ficheros cargados se repartirán en canales consecutivos.

3

Descripción de los principales parámetros del editor de muestras.

4

Descripción de los principales parámetros del sintetizador TS404.



5

Procedimientos para la asignación y edición de efectos.

TRUCO

Para modificar los parámetros de todas las notas a la vez de una pista, mantenemos pulsada la tecla "Ctrl" mientras dibujamos con el lápiz.

comportamiento es el mismo para los dos, la diferencia estriba en el tipo de sonido que reproducen. Los canales de muestras pueden reproducir ficheros de sonidos en formato .WAV, .XI (Fast Tracker), DS, (DrumSynth) y .SYN (SimSynth) y los canales TS404 reproducen sonidos creados con el sintetizador de bajos TS404.

Para insertar un nuevo canal elegimos la opción *Channel / Add one* y el tipo de canal ("Sampler"o "TS404"). También podemos hacerlo en el menú flotante que aparece al pulsar con el botón derecho del ratón sobre el botón que muestra el sonido en cada pista. En el mismo menú, también es posible realizar operaciones de clonado y borrado de canales, así como funciones de edición básicas.

MODIFICANDO SONIDOS

Al pulsar sobre el botón se abrirá una nueva ventana llamada "Channel settings" en donde podemos modificar todos los parámetros del sonido.

EDITANDO MUESTRAS

En la ventana "Channel settings", la pestaña "SMP" corresponde a los parámetros de una muestra. Desde ahí podemos cargarla, crear una nueva, modificarla o grabar la nueva muestra modificada. En la figura 3 se describe este panel.

Pulsando sobre la gráfica de la muestra la reproducimos y haciendo clic con el botón derecho del ratón, mostramos las opciones para cargarla, crearla o grabarla.

Para modificar su sonoridad tenemos varios parámetros. En primer lugar, disponemos de tres filtros ("FX1", "Sine FX" y "FX 3"), los cuales se utilizan para cambios de volumen y modulación de la onda. Además, disponemos de un amplificador ("Amp"), reverberación ("Reverb"), frecuencia de corte ("Cut"). También es posible modificar el ataque ("Att"); es decir, acentuación del comienzo y el decaimiento o fade ("Dec"). En "Reverse" cambiamos el orden de la muestra y las luces rojas "A" y "B" conmutan entre el canal izquierdo y el derecho del estéreo.

EDITANDO SONIDOS DE LA TS404

Si el canal seleccionado contiene un sonido TS404 la ventana *Channel settings* adquiere otro aspecto. En ella, se muestran todos los parámetros propios de un sintetizador (pestaña TS404).

Básicamente se trata de un sistema de síntesis para líneas de bajos. Consta de dos osciladores (OSC1 y OSC2), que son mezclados en el apartado "OSC 1+2". También podemos modificar la envolvente del sonido mediante los parámetros "Att" (Attack), "Dec" (Decay), "Sus" (Sustain) y "Rel" (Release).

Disponemos, además, de 4 filtros (LP12, LP24, BP y HP) en el apartado "Filter" y de un oscilador (LFO) para hacer oscilar el tamaño del primer oscilador o los filtros. Y por último, podemos aplicar a toda la onda distorsión (Dist) y eco (Delay).

A través de la pestaña "PITCH" podemos modificar el tono de la onda con valores absolutos ("Middle note") o relativos ("Fine") (Ver Fig. 4).

AÑADIENDO Y MODIFICANDO EFECTOS A LAS PISTAS

Para ambos tipos de canales es posible la aplicación de múltiples efectos. Éstos se eligen y asignan en la ventana *Song settings*, la cual se abre al pulsar con el botón derecho del ratón sobre las luces "FX1" hasta "FX8" de la ventana *Channel settings*. Es posible mezclar hasta 4 efectos a la vez para un canal de efectos. Los elegimos pulsando sobre la pequeña flecha invertida junto a "(none)" en "Select" o "Favorites" y lo activamos o desactivamos pulsando en el pequeño "enchufe" de la derecha. Una vez elegido el efecto, aparecerá una nueva ventana que muestra todos los parámetros de dicho efecto (Ver Fig. 5).

Funciones 3D (II). Texturizado de objetos 3D y brushes

La creación de un modelo 3D empieza con su modelado y termina definiendo su aspecto final mediante la aplicación de texturas.

Como recordaremos, una textura es una imagen 2D, diseñada con una aplicación de dibujo que cubre un objeto 3D, dándole a éste su apariencia definitiva. Blitz3D permite crear texturas, cargarlas de disco o modificarlas. En este número aprenderemos todo lo necesario para texturizar nuestros modelos y pintarlos con el uso de *brushes*.

CREANDO TEXTURAS

Se pueden crear texturas manualmente desde programación o bien a través de una imagen.

FICHEROS DE IMÁGENES COMO TEXTURAS

Este primer método es el más sencillo y consiste en crear la textura a partir de un fichero de imagen (.BMP, .PNG, .JPG, .PCX, o .TGA) Se utiliza la función:

```
LoadTexture
(Nombre_fichero_imagen
[,flags])
```

El parámetro "flags" o banderas permite la aplicación de efectos a la textura una vez cargada. Estos efectos se pueden combinar sumando banderas. En la figura 1 se muestra un esquema con las diferentes banderas y su efecto asociado. En el ejemplo:

```
Textura= LoadTexture
("C:\mi_textura.png",3);(1+2)
```

La variable "Textura" manipulará la textura creada a partir de la imagen "mi_textura" que se ha cargado con color (bandera 1) y canal alfa (bandera 2) (1+2=3).

1: COLOR (por defecto)

2: CANAL ALFA (Colores alfa serán transparentes

4: MÁSCARA (Las áreas con color no se dibujarán)

8: MIPMAPPED (Textura con bajo detalle y suavizado a grandes distancia)

16: UNIR COORD. TEXT. HORIZONTAL (U) (Evita el enrrolamiento de la textura)

32: UNIR COORD. TEXT. VERTICAL (V) (Evita el enrrolamiento de la textura)

64: MAPEADO DE REFLEXIÓN ESFÉRICA (Mapeado de entorno)

1

Lista de los parámetros "flags" posibles para la carga de texturas.

TEXTURAS ANIMADAS A PARTIR DE IMÁGENES

Un claro ejemplo de cómo B3D transforma las imágenes contenidas en ficheros a texturas lo encontramos en la función "LoadAnimTexture". Este comando permite la creación de texturas animadas por medio del encadenamiento de imágenes (frames) contenidas en un fichero (Ver Fig. 2 y "ejemplo1.bb").

Para crear este tipo de textura es necesario tener en un mismo fichero, y de forma consecutiva, las diferentes imágenes que forman la secuencia de la animación. Una vez cargada la imagen con "LoadAnimTexture", B3D separa la secuencia en fotogramas independientes y asigna la animación como textura.

TRUCO

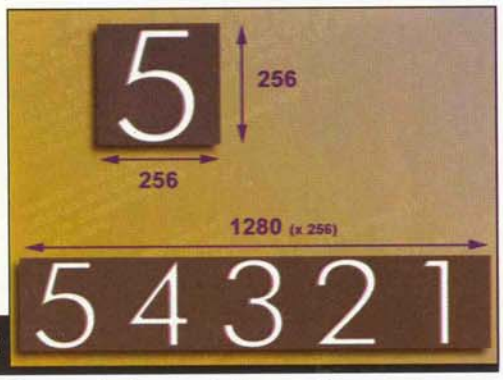
El mejor formato para utilizar texturas en Blitz3D es .PNG. Ocupa poco espacio en disco al igual que .JPG, comprime con calidad parecida a .BMP y admite canal alfa (transparencia).

NOTA

El formato .JPG no soporta canal alfa, mientras que .PNG y .TGA sí. Si utilizamos un formato sin canal alfa, Blitz3D lo emulará creando uno propio por medio de valores de color.

NOTA

El consumo de memoria al cargar texturas no depende del formato utilizado. Si cargamos texturas en .JPG no significa que ocuparán menos memoria por estar comprimidas. Ocuparán lo mismo que si están en formato .BMP. La explicación de esto es que Blitz3D utiliza su propio formato; es decir, cuando se carga una imagen para convertirla en textura, independientemente del formato, Blitz3D convierte esa imagen en una textura con su propio formato interno.
La única diferencia que encontramos en el uso de diferentes formatos está en el tiempo de carga. Por ejemplo, una textura en formato .BMP tardará más tiempo en cargarse que la misma en .JPG ya que, al no estar comprimida, el fichero es mayor.



2

Esquema de cómo construir una textura animada con un programa de dibujo.

```
LoadAnimTexture (TexturaS,
Flags, Tamaño horizontal
fotograma, Tamaño vertical
fotograma,
   Primer fotograma, Número
   de fotogramas totales)
```

Al igual que en la función "LoadTExture", es posible aplicar los mismo efectos por medio de "Flags". Es importante definir el tamaño que tendrá cada fotograma exactamente igual a como lo hayamos dibujado; es decir, si la secuencia de imágenes la forman cuadros de 64 x 64 debemos colocar en "Tamaño horizontal" y "Tamaño vertical" un 64. El "primer fotograma" corresponderá al 0. Si en el "Número de fotogramas totales" ponemos un valor inferior al número de imágenes contenidas en la secuencia del fichero, B3D rellenará los huecos con blanco. Por ejemplo, disponemos de un fichero en donde hemos dibujado una secuencia de imágenes para la textura del agua. Este fichero contiene 10 fotogramas (0-9) de tamaño 64 x 64:

```
Textura_animada=
LoadAnimTexture
("Secuencia.png",
3,64,64,0,9)
```

Es importante tener en cuenta que en el ejemplo anterior la imagen del fichero original debe tener un tamaño total de 640 X 64 (10 Cuadros de 64 = 640 x 64 de altura). Si no fuera así, ocurriría un error.

CREANDO TEXTURAS MANUALMENTE DESDE PROGRAMACIÓN

Otra posibilidad muy flexible e interesante que permite B3D es la creación de texturas a partir de cero y dibujarlas a través del buffer de texturas "TextureBuffer".

```
TextureBuffer (Textura
[,fotograma])
```

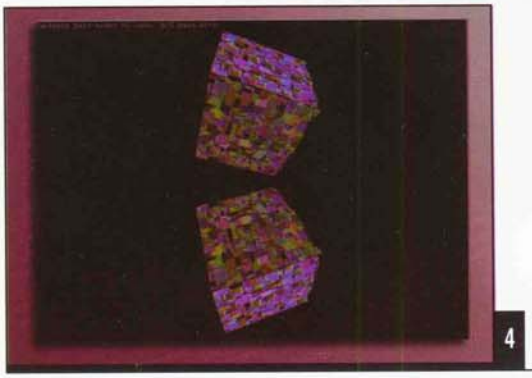
Para crear una textura nueva utilizamos la función "CreateTexture":

```
Nueva_Textura=CreateTextura
(Tamaño horizontal en píxel,
Tamaño Vertical [,flags]
[,fotogramas])
```

Las opciones de efectos aplicados según "flags" son las mismas que en las funciones anteriores (ver Fig. 1) y la opción "Fotogramas" indica el número de fotogramas que tendrá la textura si así fuese. Por defecto su valor es 1.

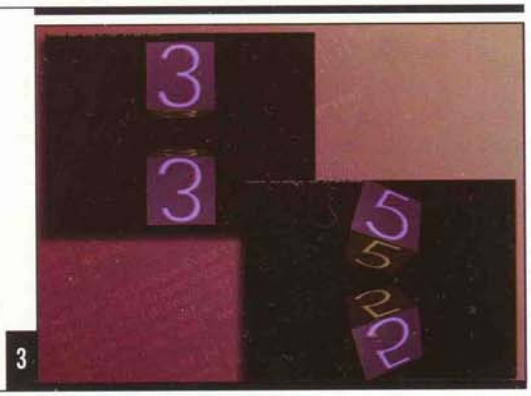
Podemos crear el tamaño de texturas que queramos. El único límite lo impone la tarjeta gráfica. De todas formas, si la tarjeta no soportara el tamaño que hayamos especificado, B3D asignará automáticamente el más cercano disponible. También, normalmente se usan texturas cuadradas con tamaños múltiplos de dos iguales; es decir, 256 x256, 64 x64. Aunque es lo más normal –ya que lo soportan todas las tarjetas-, es posible asignar tamaños diferentes como 256 x 128 o 512 x 64, siempre y cuando los valores sean múltiplos de 2.

Para dibujar graficos 2D en la textura primero debemos activar el búfer de textura: *SetBuffer TextureBuffer* Seguidamente, dibujaremos la imagen y, por último, la copiamos al búfer de texturas. Se puede ver el funcionamiento de este proceso en los ejemplos "ejemplo2.bb", "ejemplo3.bb" y "ejemplo4.bb".



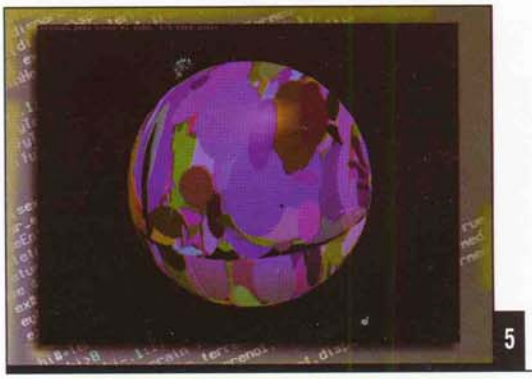
4

Textura aplicada a un cubo, la cual ha sido previamente dibujada desde programación en el búfer de texturas.



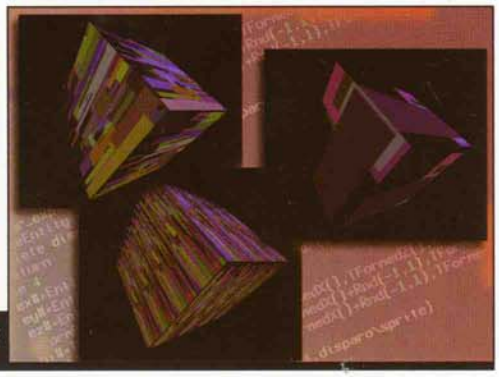
3

Captura sobre un ejemplo de aplicación de texturas animadas.



5

Ejemplo del proceso de dibujar en el búfer de textura en tiempo real.



6

Es posible escalar las texturas en cualquier dirección.

Si lo que deseamos es dibujar gráficos 3D, no podemos hacerlo directamente en el búfer de texturas. Tenemos primero que dibujar en el "BackBuffer" y luego con la instrucción "CopyRect" copiarlos al "TextureBuffer".

MANIPULANDO TEXTURAS

Disponemos de multitud de comandos para el manejo de texturas, desde escalarlas hasta cambiar sus coordenadas en el modelo.

Para cambiar el tamaño de una textura utilizaremos la instrucción "ScaleTexture":

```
ScaleTexture Textura,
Escalado U (Horizontal),
Escalado V (Vertical)
```

(Ver "ejemplo5.bb").

Para rotar una textura aplicada a un modelo utilizaremos "RotateTexture":

```
RotateTexture Textura, Ángulo
```

(Ver "ejemplo6.bb").

Esta instrucción es de acción inmediata y resulta muy útil para aplicar efectos a los modelos. Suele ser muy utilizada en sistemas de partículas.

NOTA

Una vez aplicada la textura al modelo podemos borrarla de la memoria con el comando "FreeTexture Textura". Una vez borrada no podrá ser usada de nuevo; sin embargo, el modelo seguirá texturizado.

SCROLL DE TEXTURAS

Hay un último comando que nos permite realizar desplazamientos de la textura aplicada a un objeto 3D, muy útil para realizar scroll de texturas y simular movimiento (Ver "ejemplo7.bb"). Por ejemplo, está muy extendido su uso en texturas utilizadas para simular la superficie del agua. En el "ejemplo8.bb" se muestra la aplicación de este sistema para simular un río de lava.

MEZCLANDO TEXTURAS; "BLENDING"

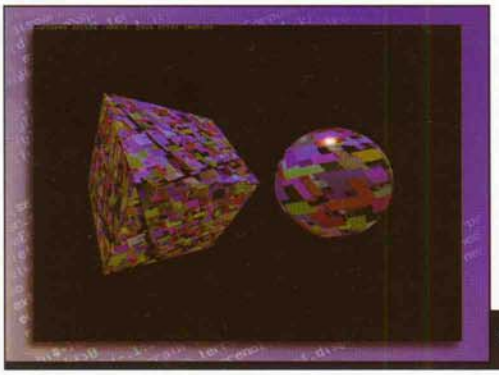
La técnica "blending" establece la forma en que la textura se mezclará con el objeto u otra textura (en caso de multitexturizado). Es posible mezclar hasta 8 texturas según la tarjeta gráfica. La textura número 0 se mezclará con los polígonos del objeto aplicado, la textura 1 con la textura 0 y así sucesivamente.

En B3D es posible aplicar esta técnica mediante la instrucción "TextureBlend": (Ver "ejemplo9.bb")

```
TextureBlend Textura,
Modo de mezcla
```

El parámetro "Modo de mezcla" determina el modo "blending" que B3D aplicará:

- **Valor 0:** no se aplicará ninguna textura.
- **Valor 1:** no se realiza mezcla alguna o sólo el canal alfa si lo hubiera.
- **Valor 2:** se multiplica la mezcla. Es el valor por defecto.
- **Valor 3:** se suma la mezcla; es decir, en multitexturizado, se suma una textura con otra.



7

Ejemplo de rotación de texturas sobre meshes.

CREACIÓN DE *BRUSHES*

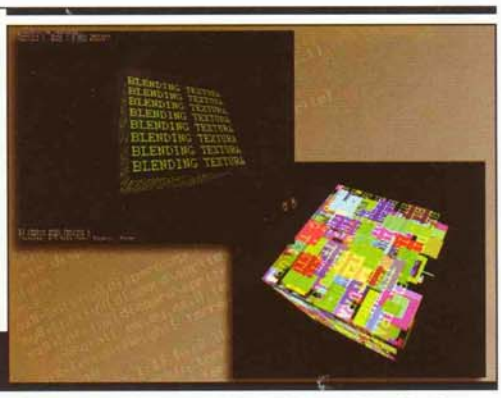
Un *brush* o pincel es un grupo de propiedades (color, brillo, textura, etc.) que pueden ser aplicadas a una entidad, superficie o *mesh*. Los *brushes* se utilizan, por ejemplo, para cambiar el aspecto de algunos polígonos del modelo, ya que, con funciones de entidad como "EntityColor", cambiaremos el color el objeto completo.

Para asignar o pintar con un *brush* a un *mesh* utilizamos *PaintMesh Mesh, Brush*.

Para hacer lo propio con una entidad: *PaintEntity Entidad, Brush* y para pintar una superficie: *PaintSurface Surface, Brush*.

8

Un ejemplo práctico sobre la técnica de scroll de texturas para simular el fluir de un río de lava.



9

Ejemplos de la técnica "blending" (mezcla) aplicada a texturas.

CREANDO *BRUSHES* MANUALMENTE

Para crear un *brush* desde cero utilizaremos la instrucción "CreateBrush":

```
Nuevo_Brush= CreateBrush
( [componente color rojo#]
[, verde#][,azul# ] )
```

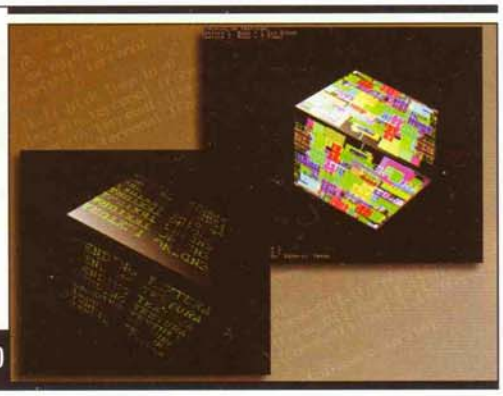
Podemos crear un *brush* nuevo y posteriormente asignarle una textura con el comando "BrushTexture":

```
BrushTexture Brush, Textura
[,fotograma de la textura]
[, índice de la textura]
```

Ejemplo:

```
Textura=LoadTexture("mi_foto.png")
Nuevo_Brush=CreateBrush()
BrushTexture Nuevo_Brush, Textura
```

Es posible asignar hasta 4 capas (0-3) de texturas por cada *brush* por lo que se pueden obtener muchos más efectos.



10

Mediante el "blending" se pueden mezclar texturas de diversas maneras.

CREANDO *BRUSHES* A PARTIR DE FICHEROS DE IMÁGENES

Con la instrucción "LoadBrush" podemos cargar un fichero de imagen, crear una textura con él y asignarla posteriormente a un *brush*:

```
Nuevo_Brush= LoadBrush
(Imagen[,Flags][,Escala U]
[,Escala V]
```

Las opciones "Flags" son las mismas que las de las texturas (ver Fig. 1).

Además, podemos definir directamente el escalado que tendrá la textura aplicada al *brush*.

APLICANDO EFECTOS A LOS *BRUSHES*

Al igual que ocurre con las texturas, es posible aplicar ciertos efectos a los *brushes* como brillo, color, etc.

Para asignar un color a un *brush* utilizaremos:

```
BrushColor Brush, Componente
color rojo#, verde#, azul#
```

Se puede obtener un brillo más intenso en ciertas partes del modelo donde la luz incide por medio del comando:

```
BrushShininess Brush,
Cantidad de brillo# (0-1)
```

También es posible utilizar "blending" en los *brushes* con:

```
BrushBlend Brush,
Modo de mezcla.
```

11

Mediante la aplicación de efectos a un brush se pueden obtener resultados muy interesantes.

Los modos de mezcla son: el 1 canal alfa, 2 multiplicar y 3 sumar.

Hay una instrucción muy interesante que es "BrushAlpha", y se puede utilizar para ocultar una entidad en el lugar de "HideEntity", en el caso de que se quiera seguir detectando por el sistema de colisiones.

Por último, disponemos de otro comando para la aplicación de efectos extras:

```
BrushFX Brush, Efecto
```

Los parámetros de efectos pueden ser sumados para combinar efectos.

Los valores posibles para "Efecto" son:

- **0:** Por defecto (ningún efecto).
- **1:** Brillo.
- **2:** Usar "Vertex Colors" en vez de "Brush Color".
- **4:** Sombreado plano.
- **8:** El efecto de niebla no afectará al modelo.
- **9:** Desactiva la ocultación de los polígonos traseros. Se muestran todos los polígonos en modo "Wireframe" (alámbrico) (Ver "ejemplo10.bb").

En el próximo número...

... estudiaremos el sistema de animación y las técnicas más recomendadas.

Cómo manejar niveles BSP

Empezamos una serie de tutoriales relacionados con el desarrollo y programación de juegos. Dedicaremos este espacio para construir rutinas útiles.

Además, responderemos a cuestiones que quedaron pendientes durante el curso y ampliaremos otras que, por su interés, merecen un estudio más a fondo. Para empezar, estudiaremos una característica muy interesante contenida en Blitz3D: el manejo de niveles BSP.

MUNDOS BSP

La nueva posibilidad que Blitz3D, a partir de su versión V. 1.76, brinda para el uso de niveles BSP, posibilita la creación de una manera sencilla de juegos al estilo Quake. Antes de seguir, recordemos qué significa BSP y en qué consiste esta técnica:

BSP son las siglas de *"Binary Space Partitioning Tree"* – árbol binario para la partición de espacio- y es, básicamente, una forma de organizar objetos dentro de un espacio. Esta técnica aplicada a un entorno 3D coloca los objetos en una estructura de árbol que es seguida por el programa de forma ordenada y decide qué polígono se debe mostrar y cuál no. Es un buen método para eliminar del campo de visión áreas ocultas, con lo que se aumenta en gran medida el rendimiento total.

Esencialmente, todo el trabajo que representa el manejo de árboles BSP lo resuelve Blitz3D para el usuario mediante su filosofía de entidades. Esto quiere decir que se puede cargar un nivel BSP completo y asignarlo a una variable como si de un modelo 3D se tratara. Así que, al ser una entidad, se le pueden aplicar los comandos básicos de éstas como posicionamiento, escalado o rotación, así como todas las funciones completas de colisión.

NOTA

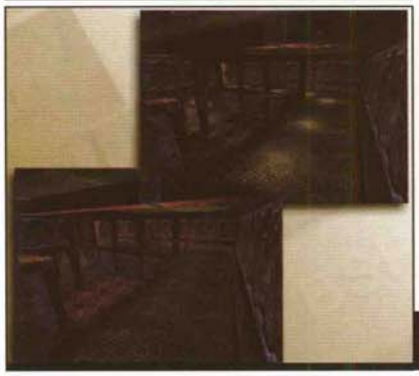
Las cualidades del manejo de niveles BSP se encuentran disponibles a partir de la versión 1.76 de Blitz3D.

```
LoadBSP ( "Nivel.bsp"
[, ajuste gamma] [,madre])
```

La opción "ajuste gamma" nos permite modificar la intensidad de luz del mapeado de luces del nivel BSP. El valor por defecto es 0 y va desde 0 hasta 1.

Ahora bien, hay ciertos puntos que debemos tener presentes a la hora de utilizar modelos BSP:

- **1.** Los modelos BSP no pueden ser texturizados, pintados, coloreados, etc... desde programación.
- **2.** Las texturas utilizadas en un nivel BSP deben ir en el mismo directorio que el modelo.
- **3.** Los modelos BSP no son iluminados por las luces ambientales del Blitz3D. Así que debemos utilizar la iluminación propia para el modelo BSP ("BSPAmbientLight").
- **4.** Si aplicamos "AmbientLight" normal del Blitz3D, éste se sumará al "AmbientLight" del nivel BSP, de tal manera que podemos jugar con estos dos valores para obtener algún tipo de efecto de iluminación.

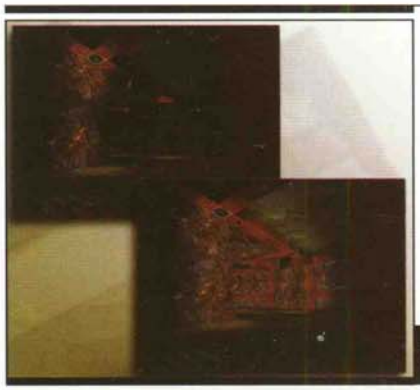


Se muestra la activación de "LightMapping". Y en la imagen inferior "Vertex Lighting".

```
BSPAmbienLight variable que
contiene el modelo BSP, Valor
de Rojo#, Valor de Verde#, Azul#
```

- **5.** Es posible definir qué sistema de iluminación afectará al mundo BSP entre mapeado de luces (LightMapping) o iluminación de vértices (Vertex Lighting). Este último suele ser más rápido en algunas tarjetas gráficas pero no ofrece la misma calidad que un mapeado de luces.

```
BSPLighting variable que
contiene el modelo BSP, tipo
de iluminación (True/False)
```



Una zona sin iluminar (imagen superior) y la misma iluminada con el uso de la linterna (imagen inferior). Sistema "LightMapping" activado.

3

Una zona sin iluminar (superior) y la misma iluminada con el uso de la linterna (inferior). Sistema "Vertex Lighting" activado.

Si en el parámetro "Tipo de iluminación" colocamos "True" se utilizará el mapeado de luces; si por el contrario utilizamos "False" elegiremos la iluminación de vértices (Fig. 1).

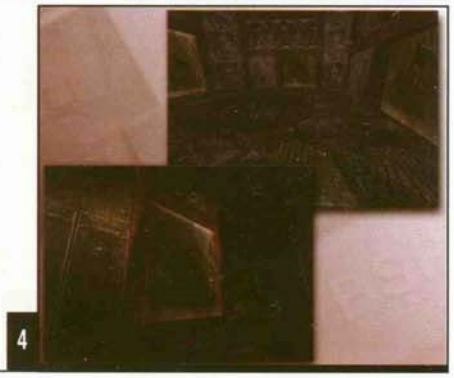
Ejemplo:

```
...
Nivel = LoadBSP("nivel1.bsp",0.5)
BSPAmbientLight Nivel,200,150,150
BSPLighting Nivel, True
...
```

UN EJEMPLO PRÁCTICO

INTRODUCCIÓN

Incluimos en el CD un ejemplo de cómo manejar niveles BSP llamado "BSP.bb y BSP.exe". En él utilizamos colisiones, iluminación y cómo implementar un sistema de puertas automáticas. A continuación, haremos una descripción del programa para esclarecer el uso de niveles BSP.



4

Imágenes de las puertas automáticas implementadas en el nivel BSP.

El ejemplo adjunto admite la posibilidad de navegar por el interior de un nivel BSP mediante una cámara y el uso del ratón y los cursores. También es posible activar o desactivar una linterna para estudiar las posibilidades de iluminación en tiempo real, así como la elección de los dos modos de iluminación posibles: "LightMapping" y "Vertex Lighting". Además, se incluye un sistema que utiliza una estructura de datos para colocar puertas en diferentes lugares del nivel, las cuales se abren y cierran automáticamente al ser tocadas.

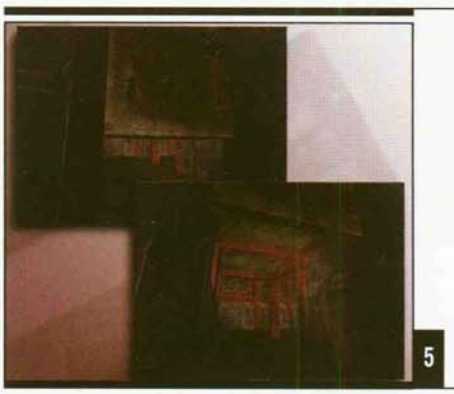
DESCRIPCIÓN BREVE DEL PROGRAMA

Después de definir las constantes para el sistema de colisiones y las variables necesarias, creamos la cámara que utilizaremos como entidad principal (Ver comentario"Define camara").

Para crear la linterna, utilizamos una luz de tipo "Spot" con un cono de luz de 60 grados y dependiente de la cámara: *Linterna=CreateLight(3,camara): LightConeAngles Linterna,0,60* (Ver Fig. 2 y 3).

Creamos una puerta a partir de un cubo escalado. Esta entidad nos servirá luego para hacer las copias de las demás puertas. Seguidamente, definimos una matriz (contendrá las posiciones y el estado de cada puerta) y su estructura correspondiente (Ver "Define puertas"). Después de cargar y definir el nivel BSP, creamos el mapa de colisiones y llamamos a la función para crear las puertas *Crear_Puertas()*. En esta función, simplemente, añadimos una nueva puerta a la estructura y rellenamos los campos a partir de la matriz de datos (Ver "Función Crear Puertas") (Ver Fig. 4).

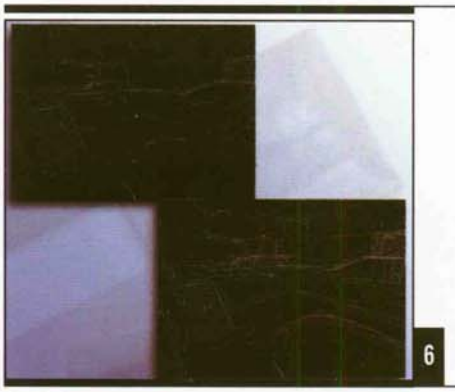
Dentro del bucle principal destacamos el procedimiento para detectar y guardar qué puerta hemos tocado (Ver "Detecta puertas"). A continuación, abrimos o cerramos las puertas dependiendo de la orden contenida en las variables *SW_Abrir* y *SW_Cerrar*. Apuntamos que la variable auxiliar *Aux_Puerta* contiene la entidad *puerta* que hemos tocado y que pasaremos luego a las funciones de apertura y cierre (Ver Fig. 5).



5

Al chocar con las puertas, éstas se abren hacia arriba y al segundo se vuelven a cerrar.

Para terminar, el resto de instrucciones se refiere a la activación de la linterna y de los diferentes modos de iluminación y de todo el control por el usuario de la cámara (Fig. 6).



6

Arriba, modo en alambre (wireframe) con "LightMapping" y abajo el mismo modo pero con "Vertex Lighting".

En el próximo número...

... explicaremos la manera de implementar los diferentes sistemas de creación de cielos para nuestros entornos.

Juegos de estrategia (II). Estrategia en tiempo real

Continuamos con los juegos de estrategia, conociendo la historia de un género que domina prácticamente la producción de juegos en la actualidad: los RTS o juegos de estrategia en tiempo real.

La producción de títulos de este género es enorme y se cuenta por cientos. Debido a esto, sólo mencionaremos los más significativos. También debemos hacer una distinción dentro de este género según la premisa básica del objetivo del juego; es decir, si tiene o no un carácter bélico.

UN PASO DE GIGANTE

Volvamos atrás unos años y conozcamos la chispa que generó la masiva incursión de las partidas en tiempo real para un tipo de juego exclusivo de unos pocos donde las técnicas de tablero aún dominaban al PC. Allá en los comienzos de los 90, a un señor llamado *Brett Sperry*, obsesionado por buscar la diversión en los juegos de estrategia por turnos de aquel momento, se le ocurrió la brillante idea de que las posibilidades del ordenador permitirían eliminar el sistema por turnos y hacer viable un género más atractivo, fácil y sobre todo más entretenido. Nació entonces *Dune II* en 1992, de la mano de su desarrolladora WestWood Studios, el primer juego de estrategia en tiempo real de la historia. A partir de ahí, WestWood creció meteóricamente y publicó su siguiente trabajo, *Command & Conquer*. Le siguió una saga completa basada en el mundo de C & C, la cual se convertiría en la más vendida de la historia de los juegos de estrategia y construyó las bases del género: *C & C: Red Alert I* y *II* (1997 y 2000), *C&C: Tiberina Sun* (2000) (Dunell ,C&C).

Al igual que ocurrió en el género por turnos, apareció paralelamente al ambiente bélico de C&C, un título ambientado en los mundos de *Warhammer*, *Warcraft* (Blizzard, 1994), en el que combaten orcos contra humanos. Le siguió *Warcraft II* (Blizzard, 1996), que ha llegado a convertirse en un clásico.

EVOLUCIÓN TÉCNICA

Empezamos como base dando un repaso a los títulos que usan una ambientación de conflictos armados para establecer el argumento del juego.

La aparición de la estrategia en tiempo real originó la base fundamental de este genero, la Inteligencia Artificial (IA). Hasta la llegada del 3D, la única evolución que sufrían estos juegos hacía referencia sólo el apartado de la IA. De una forma u otra la base impuesta por WestWood para sus juegos era copiada hasta la saciedad, y prácticamente la interfaz de usuario o la jugabilidad eran iguales en todos los títulos: *War Wind* (*SSI*, 1996), *KKND* (Melbourne House,1997), *Earth 2140* (Topware, 1997), *Dark Reign* (Activision, 1997), etc. Aunque también aparecieron juegos originales con otro tipo de ambientación como: *Settlers* II (Blue Byte, 1996) y sus pequeños colonos; la mezcla de rol y estrategia de *Lords of Magic* (Sierra, 1997); los robots de *Z* (BitMap Brothers, 1996) y *Sindícate Wars* (BullFrog, 1996) y, cómo no, *Sid Meier´s GettysBurg* (Firaxis, 1997), basado en la Guerra Civil Norteamericana, etc. Pero en general, quizás la única diferencia radicaba en los gráficos y en la ambientación. Desde *Dune II* hasta 1997 el género no cambió

1

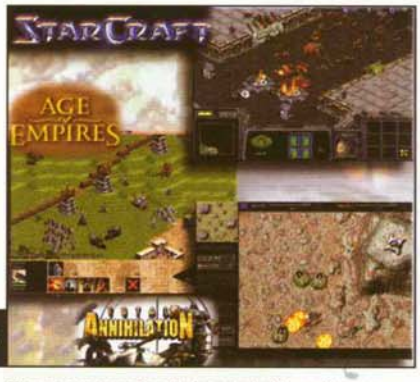
Dune II y Warcraft fueron los primeros juegos de estrategia en tiempo real de la historia.

2

La serie Command & Conquer estableció las bases del género.

3

Z y Gettysbug fueron dos ejemplos de títulos que se distinguían por su original ambientación en una época de producción masiva.



4

Tres pesos pesados de la estrategia en tiempo real: Starcraft, Age of Empires y Total Annihilation.

5

Imágenes de los primeros juegos del género en 3D.

6

Giants y Sacrifice, dos joyas de arcade – estrategia.

7

Dos apuestas sobresalientes a la estrategia espacial en 3D.

mucho. Fue entonces cuando apareció *Starcraft* (Blizzard, 1997), considerado uno de los mejores juegos de estrategia de todos los tiempos. En este mismo año irrumpe también Microsoft con *Age of Empires* que ganó el podium de los juegos de estrategias históricos. Así mismo, en este año tan prolífico apareció un nuevo concepto de jugabilidad con el título *Total Annihilation* (CaveDog,1997).

EL PASO A LAS 3D

Una vez más, la evolución técnica en los compatibles llama a la puerta de las desarrolladoras de este género ofreciendo las nuevas tarjetas gráficas y señalando hacia el 3D. La perspectiva isométrica deja paso a la tercera dimensión y fue Bungie con su juego *Mit.: The Fallen Lords* los primeros en adoptarla en Noviembre de 1997. Un nuevo mundo se abre ante la posibilidad de visualizar los combates desde diferentes puntos de vista y de poder ver con más detalles todas las unidades. Esto provoca una nueva revolución comercial que hace que 7 de cada 10 juegos sean de estrategia. Obviamente, aparecieron nuevos conceptos y sobre todo mejoras gráficas que abarcaban todos los argumentos imaginables. Sin embargo, los clásicos se mantienen y aparecen nuevas partes y expansiones adaptadas al 3D como: la tercera parte de Age of Empires, *Age of Mithology* (Microsoft, 2001), *Warcraft 3* (Blizzard, 2001), *Dark Reign 2* (Activision, 2000) o *Emperor: Battle for Dune* (2001), primera incursión de WestWood en las 3D.

Sin embargo, la isometría seguía siendo la preferida de algunas desarrolladoras y sobre todo de los incondicionales del género y aparecieron títulos realmente brillantes como *Tzar* (Infinite Loop, 2000), *Seven Kingdoms* (Interactive Magic, 1998), o *Space Clash* (Dinamic Multimedia,1999).

A partir de 1999, la mayoría de los títulos se desarrollaban en 3D y la potencia de las máquinas hacían cada vez más viable el manejo de enormes cantidades de unidades y sobre todo un mayor espectáculo gráfico. Títulos como *Shogun: Total War* (Creative Assembly / Electronic Arts, 2000) pueden mostrar incluso hasta diez mil unidades a la vez en pantalla luchando entre sí o *Warrior Kings* (Sierra, 2001) en donde es posible admirar las batallas a pie de soldado.

Aparecen también títulos que mezclan la estrategia con el más puro arcade dando un nuevo concepto al género y proporcionando una jugabilidad fresca y asequible. Destacamos *Sacrifice* (InterPlay, 2000) desarrollado por los siempre originales chicos de Shiny Entertaiment o *Giants: Citizen Kabuto* (Planet Moon/ Interplay, 2000).

Entra en juego la ambientación espacial y futurista para muchos títulos, encabezado por el fantástico *Homeworld* (Sierra, 1999), primera incursión de la estrategia espacial en las tres dimensiones con unos resultados increíbles ya que es posible controlar las batallas desde cualquier ángulo posible. Un año después, Sierra publica *Ground Control*, basado en el éxito de Homeworld, pero esta vez en tierra firme. Siguiendo los pasos de Sierra, se publican nuevos títulos ambientados en el futuro y sobre todo en el espacio como son: *Far Gate* (Super X Studios, 2001), *Conquest: Frontiers Mars* (Ubi Soft, 2001), *Star Wars: Force Commander* (Lucas Arts, 2000), *Star Trek: Klingon Academy* (14 Degrees East/Interplay, 2000), etc.

Muchos son los conceptos diferentes en torno a la estrategia en tiempo real que aún nos quedan por ver. Subgéneros como los RPS (Role Player Strategy), la construcción de ciudades o la exclusiva gestión de recursos en todos los ámbitos conocidos, quedan pendientes.

En el próximo número...

... conoceremos más sobre la historia de los juegos de estrategia en tiempo real.

Cuestionario Videojuegos 11

Preguntas

1. ¿Cómo podemos dibujar con Blitz3D una textura en un objeto 3D en tiempo real?
2. ¿Qué son los *brushes* y cómo podemos crearlos en Blitz3D?
3. ¿Cómo podemos conseguir que nuestra nave espacial acelere al pulsar la tecla del espacio y desacelere al soltarla?
4. ¿Qué instrucción utilizaremos para que, por ejemplo, la cámara apunte siempre hacia un objeto?
5. ¿Cómo podemos cambiar la tonalidad de una textura en Deep Paint 3D?
6. ¿Cómo podemos darle la vuelta a los vértices de un polígono creado en Milkshape3D?
7. ¿Cuántos tipos de canales utiliza Fruity Loops y cuáles son?
8. ¿Cómo podemos añadir un efecto a una pista en Fruity Loops?
9. ¿Cómo podemos modificar la intensidad de luz de un nivel BSP en Blitz3D?
10. Define una luz en Blitz3D para utilizarla como linterna en un nivel BSP.

Respuestas al cuestionario 10

▷ **1.** En primer lugar creamos un *mesh* y luego una superficie:

```
nuevo_mesh=Createmesh()
Superficie=CreateSurface(nuevo_mesh)
```

Seguidamente sumamos vértices y los unimos:

```
AddVertex Superficie,-1,1,0
AddVertex Superficie,1,1,0
AddVertex Superficie,1,-1,0
AddVertex Superficie,-1,-1,0
AddTriangle Superficie,0,1,2
AddTriangle Superficie,0,2,3
```

Para finalizar, actualizamos las normales:

```
UpdateNormal nuevo_mesh
```

▷ **2.** Cargando el modelo con *LoadAnimMesh*, preguntar por la parte "cabeza" (suponiendo que se nombró como "cabeza" en el modelador) y texturizarla:

```
Modelo= LoadAnimMesh("modelo.3ds")
For n = 1 to CountChildren (Modelo)
    Cabeza = FindChild (Modelo, "cabeza")
Next
TextureEntity Cabeza,Textura_cabeza
```

▷ **3.** Con la función "CreateTimer" y "WaitTimer":

```
Timer = CreateTimer(60)    ; 60 FPS
...
While Not Keydown(1)
    WaitTimer (Timer)
    RenderWorld
Wend
```

▷ **4.** En 2D, los ejes X e Y corresponden a los ejes X y Z en 3D.

▷ **5.** En primer lugar seleccionamos todo con la opción *Select all* en el menú *Select* y a continuación elegimos el tipo de mapeado UV con la opción *UV Mapping* en el menú *Tools*.

▷ **6.** Pulsando con el botón derecho sobre la capa "C" (color) situada en el menú de materiales.

▷ **7.** En primer lugar, hay que crear patrones en el modo *Pattern*. Seguidamente los pasamos al modo *Song* donde asignaremos la longitud del loop y lo crearemos a partir de los patrones. Para finalizar exportamos la canción (loop) a disco.

▷ **8.** En primer lugar, debemos crear patrones a través de los distintos canales. Una vez creados, pasamos a la sección *Playlist* en donde definimos la duración de los patrones en la canción. Una vez construida la canción ya hemos creado el loop.

▷ **9.** Por medio de puntos de inflexión colocados en la pista. Estos puntos se unirán formando una gráfica, la cual cambia la envolvente en toda la pista.

▷ **10.** Por medio de la opción *Mix Down to File* en el menú *Edit* del multipista.

Contenido CD-ROM 11

▶ AUDIO

■ Intelliscore MIDI Converter 5.0
Convierte en MIDI cualquier fichero de audio en MP3 o WAV.

■ Super MP3 Recorder 3.0.1
Grabación de sonido desde micrófono para luego aplicarle efectos.

■ Crystal Audio Engine 0.1b
Procesa señales digitales en tiempo real usando este programa multicanal.

■ Quartet X2 Music Studio 2.0
Crea tus propias melodías con este potente y completo editor.

■ Sequbeat
Programa de secuenciación para percusión con calidad de estudio de grabación.

■ Hammerhead Rythm Station 1.0
Crea interesantes loops para tus composiciones.

▶ DISEÑO 2D

■ Focus Magic
Aplica fácilmente efectos a tus imágenes con los filtros contenidos en este programa.

■ Collage Maker
Crea tus propios collages con esta excelente herramienta en pocos minutos.

■ SnagIt 6
Captura fácilmente imágenes o vídeo con un solo click usando esta sencilla aplicación.

■ VaryView 1.1
Visualizador de imágenes, para que organices todos tus archivos gráficos.

■ Graphics Converter Pro 4.0
Práctica utilidad para editar rápidamente tus imágenes y cambiarles el formato.

■ PhotoMark 1.0
Protege tus imágenes para que nadie las use indebidamente y sin tu autorización.

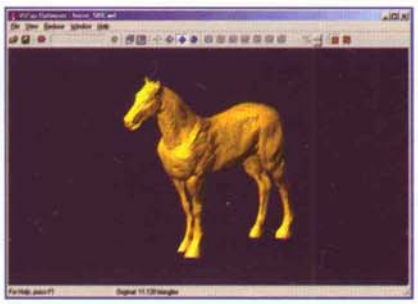
▶ DISEÑO 3D

■ 3D Exploration 1.81
Herramienta muy completa para integrar y publicar gráficos en 2D y 3D fácilmente.

■ 2D & 3D Animator 1.4
Crea, anima y edita imágenes de calidad tanto en dos como en tres dimensiones.

■ Symmetrica 3D 3.0
Poderoso programa para editar y renderizar 3D.

■ VIZup Optimizer 1.1
Herramienta de optimización 3D, para darle el máximo rendimiento a tus imágenes.



■ Meshbox 1.0
Maneja y convierte fácilmente tus contenidos en 3D.

■ Vue d'Esprit 4.1
Crea, renderiza y anima escenarios naturales ultra-realistas en tres dimensiones.

▶ PROGRAMACIÓN

■ Map Master 2.0
Toma todo el control de tus mundos virtuales gracias a esta aplicación.

■ LAN-in-a-Box 1.0.3
Con esta sencilla herramienta podrás crear partidas en red para cualquier juego en equipo.

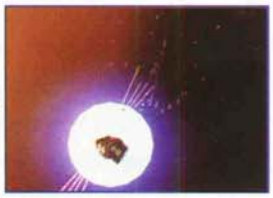
■ HexDiff 3.0
Utilísima herramienta con la que podrás controlar las versiones del código, ya que se encarga de buscar diferencias entre éstas.

■ File Substring Replacement Utility 7.0
Otra utilidad; ésta se encarga de buscar y reemplazar strings en múltiples ficheros.

■ Elegant InterFace 4.0
Personaliza tu espacio de trabajo con elegancia e imágenes usando este programa.

▶ JUEGOS

■ Homeworld
Primera incursión de la estrategia espacial en las tres dimensiones con unos resultados increíbles.



■ Age of Empires
Estupendo juego que ganó el podium de los juegos de estrategias históricos.

■ Command & Conquer
Otro de los juegos de estrategia más vendidos de todos los tiempos.

■ Zone of Fighters
Como cada semana, nuestro juego.

▶ VÍDEO

■ DivX Pro Video 5.0.2
Con este software tendrás todo lo necesario para crear tus propios vídeos en formato DivX.

■ DVD Copy Plus
Todo lo que necesitas para copiar tus películas de DVD a CD.

■ SoftReel MPEG-2 Video Decoder 1.10
Excelente filtro de decodificación para Microsoft DirectShow (ActiveMovie).

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.

Controlando al protagonista (II). Acciones

En el número anterior estudiamos los procedimientos para mover la bionave y cómo trabajar con la cámara.

En esta entrega completaremos la implementación del control del protagonista principal de nuestro juego. Fabricaremos las rutinas que harán posible las acciones de disparo, activación del camuflaje y del escudo.

```
Type Disp
  Field sprite
  Field alfa
  Field y_disparo#
  Field alcance
  Field elevacion#=20.0
  Field luz
  Field tipo_disparo
  Field retardo%
End Type
```

1

Estructura general utilizada para todos los disparos.

EL DISPARO

Todas las rutinas de control de acciones están situadas en el módulo "Jugador_principal.bb". En este módulo, además, implementaremos la creación de los disparos y su evolución en el juego.

DEFINICIÓN DEL TIPO DISPARO

Antes de continuar, es obligatorio crear previamente en el módulo "Definiciones.bb" la estructura de datos que definirá los disparos, así como todas las variables y estructuras necesarias para dar cobertura a su funcionamiento y evolución como son: las manipuladoras de entidades, controladoras de munición, explosiones, etc. (Fig. 1) (Ver Código 1).

CONTROL DEL DISPARO

La acción de disparar se produce cuando el jugador pulsa el botón izquierdo del ratón o la tecla "Ctrl" derecha. Además, es posible cambiar de arma y para ello utilizamos el botón central del ratón o la tecla "Espacio" (por si el ratón del usuario carece de tres botones).

También contemplaremos la opción de intercambiar los botones derecho e izquierdo para jugadores zurdos (Ver Fig. 2). Es de fácil programación y una opción siempre de agradecer.

```
; Cambiar botones para los zurdos
; con F10, intercambiando las
; variables "boton1" y "boton2"

If KeyDown(68) And zurdos=0
    zurdos=1
    boton1=2: boton2=1
    Delay 300
EndIf
If KeyDown(68) And zurdos=1
    zurdos=0
    boton1=1: boton2=2
    Delay 300
EndIf
```

2

Código necesario para intercambiar los botones de control del ratón para zurdos.

Código 1. Definición de los disparos

```
Type disp
  Field sprite                    ; Campo para la entidad del sprite
  Field alfa                      ; Campo para controlar el canal alfa (transparencia)
  Field y_disparo#                ; Altura del disparo
  Field alcance,elevacion#=20.0   ; Alcance del disparo y elevación máxima
  Field luz                       ; Manipulador de la luz creada en la explosión
  Field tipo_disparo              ; Tipo de disparo según arma elegida
  Field retardo%                  ; Tiempo de retardo para explosionar las bombas de retardo
End Type
; Coordenadas generales para posicionar la creación de disparos en cada cañón de la bionave.
Global x_disparoA#=12             ; Para vista en tercera persona
Global x_disparoB#=30             ; Para vista subjetiva
Type aguje                        ; Estructura para los agujeros en la pared dejado por los disparos
  Field alfa#,sprite
End Type
Type explo                        ;Estructura para las explosiones
  Field alfa#,sprite,luz
End Type
Global disparo1,disparo2,disparo3,disparo4,explosion1,sprite_agujero
Global disparo1B,disparo2B,disparo3B,disparo4B
Global carga=0, tiempo_carga=10, frecuencia_disparo=8, cambio=0, tiempo_cambio=10
```

Código 2. Condicionales para crear disparos

```
If (MouseDown(boton1) Or KeyDown(157) And carga=0
  Select Tipo_armamento
    Case 1
      Crear_disparo.disp(pivote_bionave,tipo_armamento,disparo1)
      municion=municion-1
    Case 2
      Crear_disparo.disp(pivote_bionave,tipo_armamento,disparo2)
      municion2=municion2-1
    Case 3
      Crear_disparo.disp(pivote_bionave,tipo_armamento,disparo3)
      municion3=municion3-1
    Case 4
      Crear_disparo.disp(pivote_bionave,tipo_armamento,disparo4)
      municion4=municion4-1
    End Select
  carga=1
EndIf
```

Situándonos en la función "Control_jugador_principal()" añadimos las sentencias condicionales necesarias para la creación de los disparos (Ver código 2).

La variable "carga" es utilizada como activador para el control de la recarga del arma correspondiente. Si no se utilizase, cada vez que el jugador pulsara el botón no se controlaría la creación de disparos en todas las situaciones. Así que, mientras la recarga no se haya completado, no será posible otro disparo. Hacemos un inciso antes de continuar para explicar el control de recarga. Este control lo implementamos en el módulo de control del juego "Fjuego.bb" en la función "Actualizar_juego()". También debemos utilizar un contador que controle el tiempo de carga "Tiempo_carga", el cual se inicializará cuando llegue a 0 con un valor mayor o menor controlado por la variable "Frecuencia_disparo" y el poder del arma seleccionada ("Tipo_armamento") (Ver código 3).

De vuelta al módulo de control del jugador, una vez pulsado el botón de disparo entramos en una estructura "Select .. Case" para crear uno u otro disparo según el tipo de arma seleccionada a través de la variable "Tipo_armamento".

La creación de cada disparo la realizaremos a través de la función "Crear_disparo":

```
...
  Crear_disparo.Disp
(pivote_bionave, tipo_armamento,
disparo1)
  municion=municion-1
...
```

Realmente, esta función crea un nuevo objeto "Tipo" para la colección de la estructura "Disp".

La expresión "Crear_diparo.Disp (...)" es como si escribiéramos *Nuevo elemento. Tipo disparo = New Tipo disparo*; es decir *Disparo.Disp = New Disp*.

Es una forma de utilizar funciones para la creación de objetos "Type" individualmente cada vez que se le llaman. Además, debemos pasarle los parámetros necesarios para completar la estructura. De este modo, para la entidad en sí tenemos "disparo1" (entidad) y "pivote_bionave" (entidad madre) y para clasificarla "tipo_armamento" para saber qué clase de disparo se ha creado. Es importante resaltar que utilizamos un pivote creado a partir de la bionave para que los disparos se creen a partir de ella. El procedimiento de creación del pivote lo implementamos en el momento de cargar el fichero y crear el modelo en la función "Crear_modelos()" del módulo "Funcpantaudio.bb".

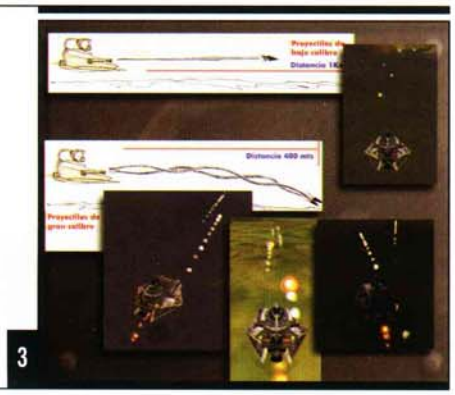
```
Function crear_modelos()
  Bionave = LoadMesh ("c:\zone
of fighters\modelos\bionave.3ds")
  pivote_bionave = CreatePivot
(bionave)
...
```

CREANDO LOS DISPAROS. FUNCIÓN "CREAR_DISPARO()"

La función "Crear_disparo()" será la encargada de producir un nuevo objeto "Type" para la colección de la estructura "Disp". Para ello, establecemos una estructura "Select .. Case" para distinguir el tipo de disparo según el armamento elegido en el argumento de función "tipo_disp".

Código 3. Control del tiempo de carga

```
If Tiempo_carga=0 Then
  carga=0
  Select Tipo_armamento
    Case 1  Tiempo_carga=frecuencia_disparo          ; Tiempo de carga estándar
    Case 2  Tiempo_carga=frecuencia_disparo+4        ; Tiempo de carga estándar + 4
    Case 3  Tiempo_carga=frecuencia_disparo*2        ; El doble del tiempo estándar
    Default Tiempo_carga=(frecuencia_disparo*2)+5    ; El doble + 5
  End Select
EndIf
Tiempo_carga=Tiempo_carga-1                          ; Decrementamos el tiempo de carga
```



3

Diseño y resultado de la munición de bajo calibre (Tipo 1, estándar) y de los misiles (Tipo 2).

En las figuras 3 y 4 se muestran los diferentes tipos de armamento utilizado en el juego.

De los cuatro tipos disponibles explicaremos sólo los proyectiles de medio calibre (estándar), ya que para los demás, el procedimiento básico es el mismo (Ver código 4).

En el caso primero ("tipo_disp=1") creamos el nuevo disparo ("disparo.disp=New disp") y a continuación asignamos todos los valores a los campos de la estructura. En la asignación "disparo\sprite = CopyEntity (entidad_disparo, pivote_bionave)" hacemos una copia de la entidad original utilizada como disparo, en este caso un "sprite" y le asignamos como "entidad madre" el pivote de la bionave. Para lograr que los disparos salgan alternativamente de los dos cañones, utilizamos el comando "TranslateEntity" aprovechando la existencia de un pivote; es decir, con esta instrucción desplazamos la coordenada X de la posición de creación del disparo con referencia al centro de la bionave "TranslateEntity disparo\sprite,x_disparoB,1,-80".

Para lograr desplazar el punto hacia cada cañón cada vez que se cree un disparo utilizamos "x_disparoB = - x_disparoB" (desplazamiento de X en 30 puntos hacia la izq. una vez y 30 hacia la dcha. la siguiente vez) (Fig. 5).

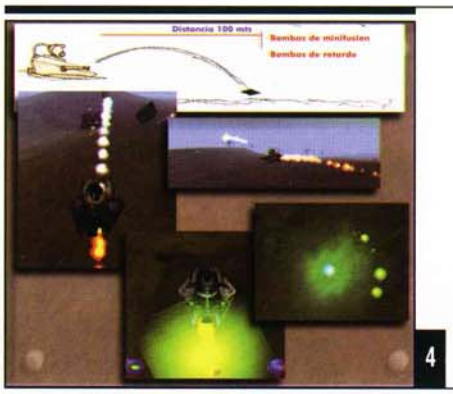
Una vez creado el disparo es importante liberarlo de la entidad madre, en este caso del pivote o centro de la bionave, para que se mueva autónomamente. Para ello, utilizamos la instrucción "EntityParent" con argumento 0 para asignar una entidad madre nula:

```
"EntityParent disparo\sprite, 0"
```

Una vez creado el disparo debemos devolver con la instrucción "Return" el nuevo objeto de la estructura creado ("Disparo").

Código 4. Creando los disparos

```
Function Crear_disparo.disp(pivote_bionave,tipo_disp,entidad_disparo)
  Select tipo_disp
    Case 1  ;Proyectiles medio calibre
      If municion<1 Return
      disparo.disp=New disp
      disparo\alfa=-90
      disparo\sprite=CopyEntity(entidad_disparo,pivote_bionave)
      disparo\alcance=300*tipo_disp  ;300 mts. calibre bajo, 600 mts
      gran calibre
      disparo\elevacion=0
      If tipo_camara=1       ; Vista subjetiva
        TranslateEntity disparo\sprite,x_disparoB,1,-80
        x_disparoB=-x_disparoB
        Else                 ; Vista en tercera persona
        x_disparoA=-x_disparoA
        TranslateEntity disparo\sprite,x_disparoA,-.5,-10
      EndIf
      EntityParent disparo\sprite,0
    Case 2 .........
    Case 3 .........
    Case 4 .........
    Return disparo
End Function
```



4

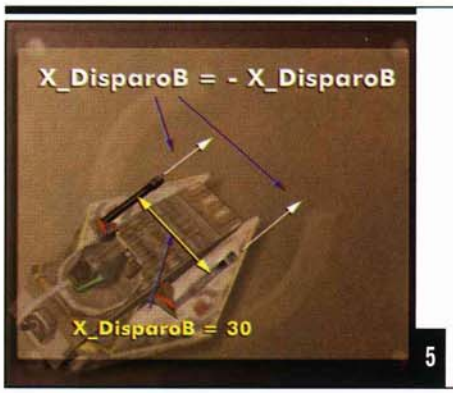
Diseño y resultado de la bomba de minifusión (Tipo 3) y de la bomba de retardo (Tipo 4).

ACTUALIZANDO LOS DISPAROS. FUNCIÓN "ACTUALIZAR_DISPARO()"

Creado el disparo, es necesario actualizar su existencia y evolución en el juego. Para ello, utilizaremos la función "Actualizar_disparo()". En ella, básicamente, desplazamos la entidad del disparo hacia su destino y controlamos su colisión con el entorno (terreno, decorado, seres, jugadores y otros disparos).

Para actualizar todos los disparos creados durante el juego es necesario recorrer la colección de datos de la estructura del tipo "Disp" y llamar a la función para cada elemento "disparo".

Este control debemos aplicarlo por cada ciclo del control de partida; es decir, en la función "Actualizar_juego()" del módulo "Fjuego.bb":



5

Esquema gráfico del funcionamiento de la función "TranslateEntity" para el disparo alternativo en ambos cañones.

6

Algunas imágenes del desarrollo de algunas explosiones.

```
For disparo.disp = Each Disp
    Actualizar_disparo(disparo)
Next
```

Llamamos a la función pasándole como parámetro el objeto "type" del disparo. De vuelta al módulo "Jugador_principal()", analizaremos la función "Actualizar_disparo()" anterior:

(Ver función: Function Actualizar_disparo (disparo.disp).

En un principio averiguamos si ha llegado al límite de su alcance para crear una explosión y eliminarla, a través del campo "alcance". Seguidamente, realizaremos todos los controles de colisión con el comando "EntityCollided". El sistema de control de colisiones es muy extenso y complejo, así que realizaremos un estudio más detallado de su funcionamiento en el próximo número.

Después de controlar las colisiones, debemos desplazar el disparo y actualizar la variación de su aspecto durante el movimiento. Utilizamos una estructura "Select .. Case" para trabajar con cada tipo de disparo. Tomemos como ejemplo el tipo estándar ("disparo\tipo_disparo=1"). En primer lugar, modificamos el tamaño del "sprite" por medio de una función seno "Sin" según su actual transparencia ("disparo\alfa") Rotamos y desplazamos el "sprite" y decrementamos su alcance.

7

Imágenes del sistema de escudo en funcionamiento.

EXPLOSIONES. FUNCIONES "CREAR_EXPLOSION()" Y "ACTUALIZAR_EXPLOSION"

Siempre que un disparo llega a su destino, bien por un impacto o bien por llegar al límite de su alcance, se produce una explosión. Como siempre, el aspecto de la explosión dependerá del tipo de disparo que la genere. Llamamos a la función "Crear_explosion()" pasándole como parámetro el objeto "type" "disparo" (*Crear_explosion(disparo)*) para poder utilizar el mismo sprite como entidad: *Function Crear_explosion.explo(disparo.disp*. Su funcionamiento es simple. En primer lugar creamos el nuevo objeto de tipo "explo" y asignamos un valor de transparencia. Seguidamente, según el tipo de disparo, damos los valores a los campos de la estructura "explo": creamos la entidad, definimos su presencia en el mapa de colisiones y posicionamos una luz y un emisor de partículas en sus coordenadas actuales. (Ver función *Function Crear_explosion. explo(disparo.disp)*).

Las explosiones debemos actualizarlas al igual que los disparos; es decir, en la función "Actualizar_juego()" del módulo "Fjuego.bb", llamando a la función de actualización (Ver función *Function actualizar_explosion (explosion.explo)*).

Básicamente, su funcionamiento consiste en aumentar y disminuir el tamaño del sprite de la explosión según el valor del campo "alfa", el cual, aumenta de 10 en 10 ("explosion\alfa=explosion\alfa+10").

ESCUDO Y CAMUFLAJE

Mediante la tecla de control izquierda activamos el escudo de la bionave y con "Alt" el camuflaje. Ambas características están regidas por una variable que regula su disponibilidad de activación. Además, por medio de "Switches" ("escudo_bionave" y "camuflaje_bionave") avisamos de su activación y también evitamos, por ejemplo, varios escudos a la vez. Estas variables nos servirán después en la función "actualizar_jugador_principal()" para llamar a sus correspondientes funciones:

```
If escudo_bionave=
1 escudo_bionave().
```

ACTIVACIÓN DEL ESCUDO

Al activar el escudo, en la función "control_jugador_principal()", creamos una esfera con ciertas características en la posición de la bionave ("escudo= CreateSphere (2, bionave)") y la introducimos en el mapa de colisiones como si fuera un disparo. Ya en la función de actualización ("escudo_bionave()") controlamos su aspecto ("TurnEntity") y el tiempo de activación por medio de las variables globales "Tiempo_escudo" y "Tiempo_escudo_total". Esta última variable es utilizada como acumulador de tiempo extra en caso de recoger bonus, etc. y para eliminar la cantidad de 100 unidades cada vez que el escudo es usado (Fig. 7).

ACTIVACIÓN DEL CAMUFLAJE

Para realizar el camuflaje de la bionave utilizaremos el comando "EntityAlpha". Haremos que la bionave quede transparente o invisible (en la vista subjetiva).

Su actividad, regulada por la función "Camuflaje_bionave()", es controlada, al igual que con el escudo, por los contadores "Tiempo_camuflaje" y "Tiempo_camuflaje_total".

En el próximo número...

... estudiaremos el sistema de colisiones que le permite actuar con el entorno.

Fabricando los elementos del juego (III)

En esta entrega daremos el aspecto definitivo a la planta carnívora Dreeck que modelamos en el número anterior y le proporcionaremos movimiento con Character FX.

TEXTURIZANDO EL DREECK CON DEEP PAINT 3D

Una vez dentro del programa, cargamos nuestro modelo "Dreeck.obj", observamos que en la ventana *Material Import* se indica la presencia de un mapeado UV. Creamos un material vacío para poder continuar pinchando en el primer cuadro de la sección *Material* en *Unknown* y elegimos un tamaño de 256 x 256 y como nuevo canal de color ("C") seleccionamos en *Add New Channel*.

Para texturizar la planta carnívora vamos a utilizar algunas opciones que todavía no hemos estudiado. No son realmente necesarias para la textura final, pero ayudarán a obtener un mejor aspecto. Se trata de utilizar más canales aparte del color, como "Bump" (protuberancias) y "Shine" (brillo).

Para activar estos canales debemos ir al panel de comandos (Panel Command) y seleccionamos en *Elements* (F7) y luego en *Layers* (Capas). Aparecerá una capa base con cinco casillas (C, B, G, S y O) que representan la paleta del material. Cuando activemos las casillas "B" y "S" nos preguntará por un tono de color, elegimos uno claro. El "Bump" proporcionará profundidad a la textura y "Shine" un aspecto más viscoso (Fig. 1).

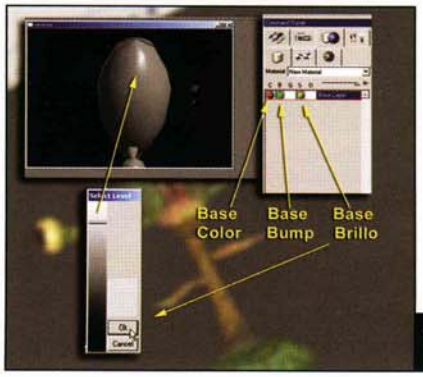
Ya estamos preparados para pintar la planta. Comenzamos por la boca. Utilizando la lupa centramos esta parte del modelo en la pantalla. Elegimos el pincel en *Presets* (F5) y para la textura la piel *Lizard Skin+* de la lista *Skin, Furs and Hair*. Seguidamente, modificamos el color y el tamaño de la piel elegida. Nos situamos en *Brush and Paint Settings* y en la pestaña *Base* de *Advanced Behavior* cambiamos la tonalidad (Hue) a verde (-215) y un tamaño en *Scale X* de 0.69 para hacer las escamas de la piel más pequeñas. Luego, más arriba, en *Brush Settings* colocamos la pluma (Feather) a 0 y una fuerza de presión (Strenght) al 100% (Fig. 2).

NOTA

Cada canal creado genera una textura independiente; es decir, si activamos el canal del color, brillo y bump, se crearán tres texturas que será posible unir por el método "blending". Evidentemente, mezclar varias texturas en un objeto supone un mayor gasto de memoria y rendimiento, a cambio, claro está, de un mejor aspecto.

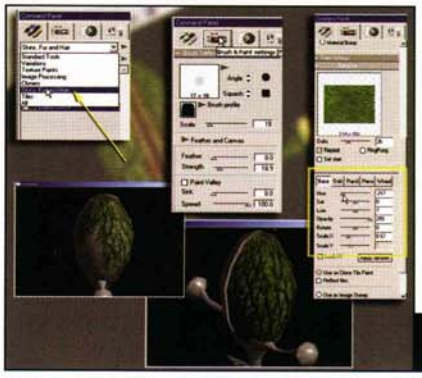
RECORDATORIO

Las letras de las casillas de las capas del material son: "C" Color, "B" Bump Mapping (profundidad), "G" Glow (resplandor), "S" Shine (brillo) y "O" Opacity (opacidad).



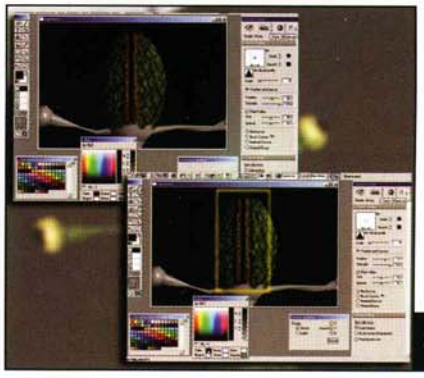
1

Se pueden obtener resultados más realistas y vistosos sumando una capa de "bump" y otra de brillo (shine).



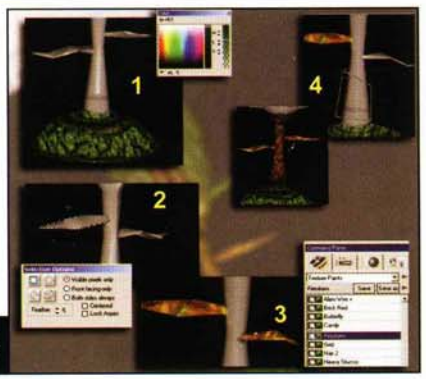
2

Utilizaremos un pincel y la textura de piel Lizard Skin + para pintar las mandíbulas.



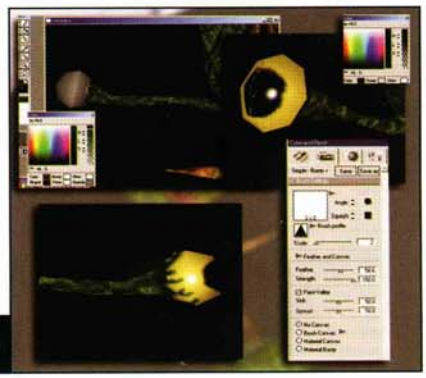
3

Podemos pintar los labios con un tono marrón y perfilarlos mediante una línea negra.



4

La herramienta de selección es muy útil para rellenar pequeñas piezas del molde como las hojas o el tallo.



5

Para dar mayor realismo se pueden pintar los nervios en tentáculos y globos oculares.

6

Unas pinceladas de tono rojizo darán un aspecto más aterrador a la planta carnívora.

Para el interior de la boca elegimos, por ejemplo, la textura *Elephant Hide* y le cambiamos el tono y la saturación para proporcionarle un color rojo a la textura. Terminamos la boca, perfilando los labios de las mandíbulas. Para ello, elegimos un color marrón en la paleta y pintamos ajustando, previamente, el pincel en tamaño y presión. Podemos aplicar más detalle al filo de las mandíbulas pintando un borde de color negro en la parte exterior del labio para proporcionar sensación de relieve (Fig. 3).

Una vez terminada la boca, pasamos a texturizar la base de la planta. Así que la seleccionamos con la herramienta de selección y rellenamos con la misma textura utilizada para la boca y con el mismo color verde. Igualmente, perfilamos los bordes de la base con el pincel con un tono de verde más oscuro y añadimos algunos trazos para simular los nervios.

El siguiente paso será pintar las hojas y el tallo. Seleccionamos una hoja y la rellenamos con la textura *FireStorm* de la lista *Texture Paints* (Fig. 4).

Seguidamente, hacemos lo mismo con la otra hoja y el tallo. Para los tentáculos oculares utilizamos la misma textura que el tronco pero con el color modificado. También en esta parte del modelo pintaremos trazos de color verde para simular los nervios. Para terminar, seleccionamos los ojos con la varita mágica de la herramienta de selección y rellenamos con un color amarillo. Luego, con el pincel pintamos de negro el iris. Podemos añadir algunos trazos en el globo ocular, en este caso también de color verde, para simular nervios o venas (Fig. 5).

Para aumentar el detalle general, pintamos algunas manchas de sangre en la boca con el pincel *Simple bump+* (Fig. 6).

Ya hemos terminado de pintar nuestro modelo. Ahora sólo nos queda salvar la textura. Si únicamente vamos a utilizar el canal del color, podemos salvar la textura pulsando con el botón derecho del ratón sobre la letra "C" y elegir la opción del menú flotante *Export Channel*. Podemos hacer lo propio para cada canal o bien a través de la opción *Save All Maps* en el menú *File*. En ambos casos elegiremos como nombre de archivo "textura_dreeck.bmp".

TRUCO

Pulsando la tecla de tabulación "Tab" ocultaremos todas las ventanas de herramientas y podremos ver el modelo terminado previamente ajustado en la ventana con "Ctrl"+ 0 (cero).

CREANDO LA ANIMACIÓN CON CHARACTER FX

Para terminar con nuestra planta carnívora vamos a asignarle algunos movimientos con Character FX. Una vez dentro del programa, cargamos el modelo "dreeck.obj" que tenemos guardado. Si no se dispone de él, es posible encontrarlo en "Extras" del CD. Una vez cargado, lo ajustamos a la pantalla y configuramos la vista de la ventana 3D en "Left". Seguidamente, le asignaremos la textura que hemos dibujado anteriormente. Seleccionamos todo el modelo y entramos en el editor de materiales (Material Editor) en el menú "Windows". Cargamos la textura "textura_dreeck.bmp" (se puede encontrar también en "Extras" del CD) y la asignamos al modelo. A continuación, elegimos la opción de visualización *Textured* en la ventana 3D para ver la planta texturizada (Fig. 7).

CREANDO EL ESQUELETO

Antes de crear el esqueleto debemos tener en cuenta qué movimientos tendrá la planta y qué partes de ella se moverán. Según el diseño, la planta será capaz de girar sobre sí misma, estirar y encoger el tallo y abrir o cerrar las mandíbulas. Con esta información, sabemos que debemos crear un esqueleto que recorra el tallo desde la base y se extienda por las mandíbulas. Necesitaremos, entonces, un punto de unión en la base, otro en medio del tallo y otro en la base de la boca. De este último, partirán dos uniones más hacia ambas mandíbulas. En la figura 8 se puede apreciar la disposición de estos puntos de unión.

Es importante colocar las uniones de abajo hacia arriba en orden. Empezaremos con la unión de la base, seguimos con la del tallo y base de la boca. A continuación, colocamos un punto en mitad de la mandíbula y otro al final. Para la mandíbula derecha, debemos partir de nuevo desde la unión de la base de la boca. Para lograrlo, debemos determinar como final de jerarquía el último punto colocado. Con esta finalidad, activaremos la casilla *IK Chain Terminator* en la ventana flotante de opciones *Tool Option* en la pestaña *Joint*. Seguidamente, con la herramienta de selección seleccionamos la unión de la base de la boca. A partir de ahora, ya podemos colocar las uniones en la mandíbula derecha (Fig. 8).

ASIGNANDO VÉRTICES

Una vez creado el esqueleto, es necesario asignar los vértices a cada unión. En este punto debemos hacer un inciso para explicar un consejo necesario para la buena implementación de este procedimiento en el caso de querer exportar la animación en formato .B3D para la versión 1.76 o superior de Blitz3D. Si una unión inferior (en la jerarquía) debe afectar a todas las uniones superiores, sólo es necesario asignar los vértices que NO son asignados a otras uniones.

Si no se siguiera este procedimiento, la animación dentro de Blitz3D no funcionaría. Para otro formato como .MD2 no hay ningún tipo de limitación.

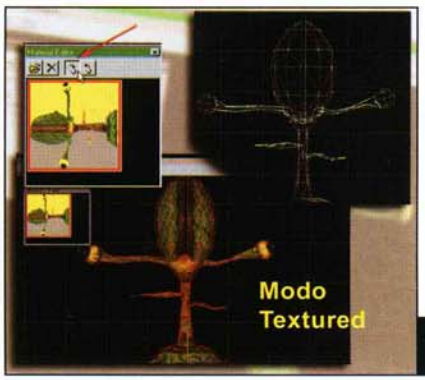
Por ejemplo, en nuestro caso, la unión del centro del tallo afectará al resto de la planta, así que lo más normal sería asignarle todos lo vértices a esta unión. Sin embargo, más arriba en la jerarquía, tenemos otras dos uniones con asignaciones, las que abren y cierran la boca. Pues bien, no es necesario incluir todos los vértices asignados a estas dos uniones. En la figura 8 se puede apreciar las asignaciones de vértices siguiendo este procedimiento. Una vez que hayamos asignado los vértices a las uniones, conmutamos al modo de animación *Animation* para comprobar los movimientos.

CREANDO LA ANIMACIÓN

Una vez que hemos comprobado que todas las asignaciones son correctas y que el esqueleto creado cumple con nuestras expectativas de diseño, pasamos a realizar el script de animación en el *Keyframer*. Antes de empezar, vamos a situar la vista en el modo *Perspective* y ajustamos el modelo con la herramienta de zoom y desplazamiento de la ventana 3D. Para que Character FX ejecute la animación a una velocidad más o menos moderada, vamos a cambiar el *Frame Rate* a 15, es decir, 15 fotogramas por segundo. Para ello, selecciona-

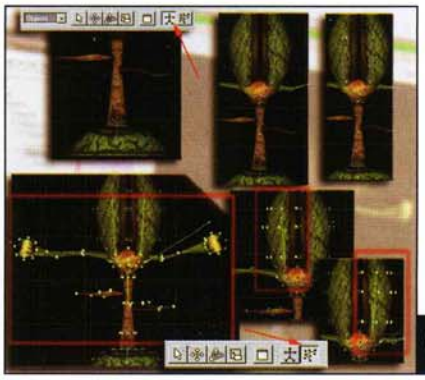
RECORDATORIO

Para sumar más fotogramas a la línea de tiempo, escribimos el número de frames en la casilla de "frame final" y pulsamos "Enter".



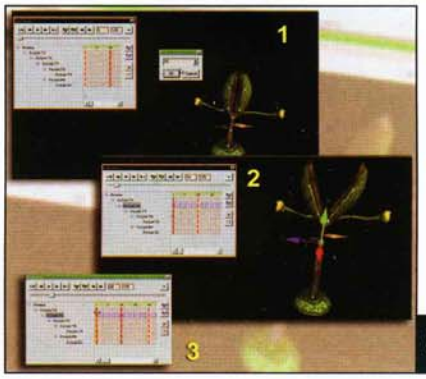
7

El primer paso para prepara nuestro modelo en Character FX es preparar la vista y texturizar el modelo.



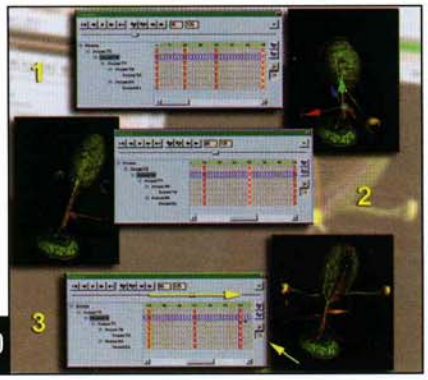
8

Debemos seguir un orden secuencial a la hora de crear la jerarquía en el esqueleto y saber qué vértices corresponden a una unión.



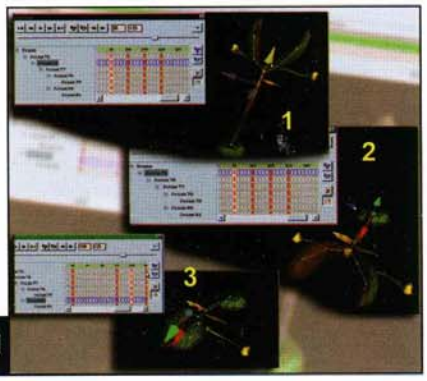
9

Copiando keyframes de un lugar a otro podemos lograr movimientos cíclicos perfectos.



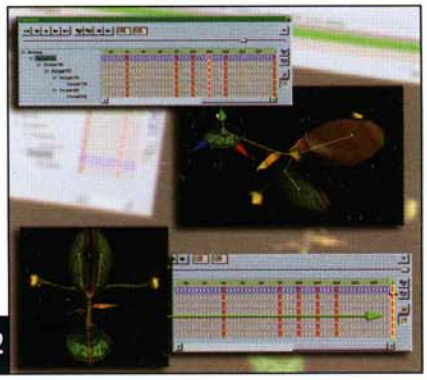
10

Seleccionando la unión del tallo giraremos toda la planta.



11

En un mismo keyframe podemos sumar movimientos de distintas uniones.



12

La velocidad de paso de un estado a otro en la animación dependerá de la cantidad de fotogramas entre keyframes.

mos la opción *Custom* en *Animation \ Set framerate* y colocamos un 15.

En la ventana *Keyframer* nos situamos en el primer fotograma y creamos un keyframe nuevo pulsando en ▣. Seguidamente, avanzamos 10 frames en la línea de tiempo y creamos otro keyframe para albergar el primer movimiento. Éste consiste en subir y bajar la planta alargando el tallo. Para realizar este movimiento, nos situamos en el último keyframe creado y con la herramienta de traslación desplazamos la unión hacia arriba pulsando en la flecha verde (eje Y). En ese mismo fotograma y con la planta desplazada hacia arriba, vamos a abrir las mandíbulas. Para ello, seleccionamos la primera unión de la mandíbula izquierda y con la herramienta de rotación la hacemos girar pulsando en la flecha roja (eje X). Hacemos lo propio con la mandíbula derecha. Una vez que hemos abierto la boca pulsamos en el botón de la llave de *Set keyframe*. Seguimos, activando el modo de copia pulsando en el icono ▣, seleccionamos todo el frame primero y lo movemos hacia el fotograma 20. La postura primera de reposo se copiará. De esta forma volvemos a mover la planta a su posición inicial perfectamente (Fig. 9).

El siguiente paso será crear el modo de búsqueda de comida que adopta la planta, consistente en observar hacia ambos lados girando el cuerpo de un lado a otro. Creamos un nuevo keyframe en el fotograma 30 y nos situamos en él. A continuación, con la herramienta de rotación, hacemos girar la planta desde la unión del tallo hasta la posición que se muestra en la figura 10. Para volver de nuevo a la posición original, realizamos la misma operación de copia, pasando el fotograma 20 al 50. En el 65 colocamos el keyframe del giro hacia la derecha y en el 80 situamos una copia de la postura del fotograma 50 (Fig. 10).

Nos queda completar el modo de ataque. Aquí, la planta se alargará exageradamente a la vez que gira hacia abajo para agarrar la víctima con las mandíbulas. Necesitaremos realizar varios movimientos en el mismo fotograma. Asignamos un nuevo keyframe en el fotograma 95 y nos situamos ahí. Para el movimiento, en primer lugar alargamos el tallo como hicimos anteriormente, pero en este caso al doble de distancia, desplazando la unión del tallo.

Luego, rotamos sobre el eje Z (flecha azul) de la unión de la base para que la planta baje hasta la altura del terreno (hacia su base) y para finalizar la acción, abrimos la boca utilizando el mismo método anterior con las mandíbulas. Para cerrar la boca, nos situaremos en el fotograma número 100, creamos un nuevo keyframe y la cerramos retrocediendo también un poco el tallo. Así, dará la impresión de que ha cogido alguna presa. Sumamos un nuevo keyframe en el fotograma 105 y nos situamos en él. Aquí, llevaremos la planta agachada rotando sobre el eje Y de la unión de la base y le abrimos la boca (Fig. 11).

Después hacemos lo mismo en el fotograma 110 y cerramos la boca. Para terminar, sumamos un nuevo keyframe en la posición 125 y copiamos ahí la posición del fotograma 1 para iniciar la animación y conseguir un ciclo perfecto.

Ya sólo nos queda salvar el modelo animado en la opción *Export* del menú *File* en el formato .MD2 o .B3D, aunque lo salvaremos en ambos (Fig. 12).

En el próximo número...

... seguiremos fabricando los elementos del juego modelando y texturizando algunos edificios.

Nuestro primer tema musical con Anvil Studio (I)

Comenzamos en este coleccionable con una serie dedicada a la creación de música para nuestros juegos mediante el secuenciador Anvil Studio. Un programa muy asequible, fácil de utilizar y con muchas posibilidades, que permite realizar cualquier tema musical a través del manejo de pistas MIDI y de muestras en formato .WAV. Anvil Studio también permite crear pistas rítmicas con nuestros sonidos de percusión favoritos y manejar librerías de sonidos sintetizados.

Es ideal para trabajar nuestros temas musicales en pistas MIDI incluso si no disponemos de ningún teclado maestro externo o sintetizador conectado al ordenador, ya que dispone de un compositor ("composer") para crear los temas nota a nota.

PRIMERA TOMA DE CONTACTO

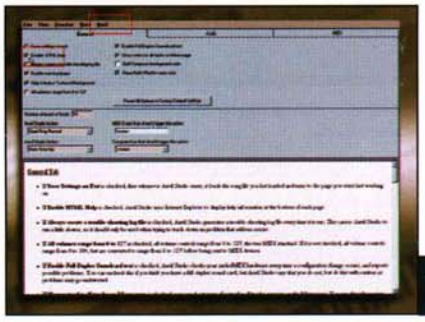
En esta entrega vamos a tener una primera toma de contacto con el programa para ir conociendo sus habilidades y facilidad de uso.

Cuando lo ejecutamos por primera vez se nos muestra interactivamente un fichero de ayuda junto a la interfaz del programa. De momento, para tener más despejada la interfaz, vamos a ocultar el fichero html de la ayuda. Elegimos la opción *View / Options*. Una vez dentro de la ventana de opciones deseleccionamos la opción *Enable HTML help* y pulsamos en *back*. Por defecto, el programa nos sitúa en la ventana *Mixer* (Fig. 1).

En esta sección es donde crearemos, editaremos y mezclaremos las distintas pistas de una canción.

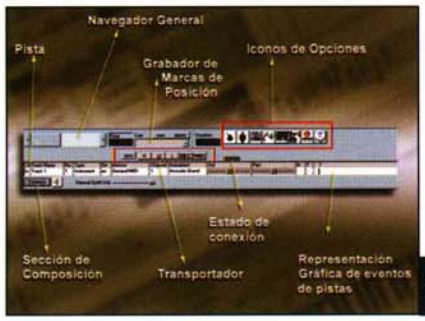
En Anvil Studio podemos trabajar con distintos tipos de pistas de manera independiente y que serán mezcladas en el mezclador (*Mixer*). Disponemos de pistas MIDI o de instrumentos, pistas de audio y pistas de ritmos. Disponemos de un transportador para reproducir y grabar y un navegador para recorrer las pistas. Disponemos también de un grabador de marcas de posición para colocarnos en diferentes puntos definidos de la canción. Asimismo, unos iconos nos permiten activar o desactivar ciertas características del programa. En la figura 3 se muestra la descripción de estos iconos.

Por defecto, sólo se muestra una pista (track), cuyos elementos pasamos a describir a continuación empezando por la izquierda. A principio de pista encontramos un cuadrado pequeño de color verde. Este color nos indica el estado de la pista que cambiará dependiendo de la casilla *On*. A continuación, en *Track Name*, podemos nombrar la pista pulsando sobre *Track 1*. Con *No* se indica el número de la pista y con *Type* a qué clase pertenece. Por defecto, aparece una pista de tipo MIDI (*instrument*), la cual podemos cambiar pulsando sobre *instrument* y eligiendo entre *Rhythm* o *Audio* (más adelante trabajaremos con cada una de ellas). En *On* definimos el estado de la pista entre *mute* (no sonará), *solo* (se mutarán las demás pistas menos la seleccionada) y *on* activa. En el apartado *Device* seleccionamos el dispositivo sonoro y dependerá del tipo de pista seleccionado. Así, por ejemplo, en una pista de tipo *instrument* aparecerá el sintetizador de sonidos correspondiente a la tarjeta en uso. Por defecto utiliza el *General MIDI*, apareciendo la lista de sonidos compatible MIDI. En el apartado *Channel* se seleccio-



1

Método para ocultar las pantallas de ayuda de la ventana de trabajo Mixer.

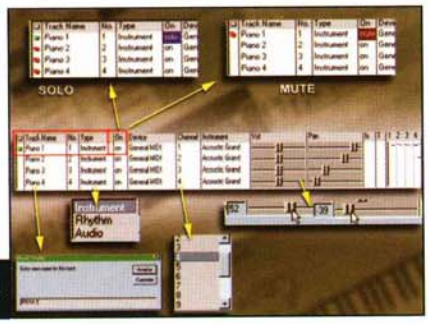


2

Descripción de los elementos del mezclador Mixer.

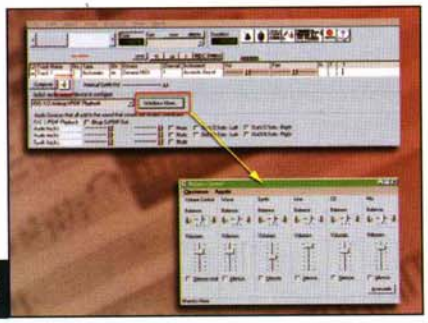
3

Descripción de los iconos de opciones del mezclador.



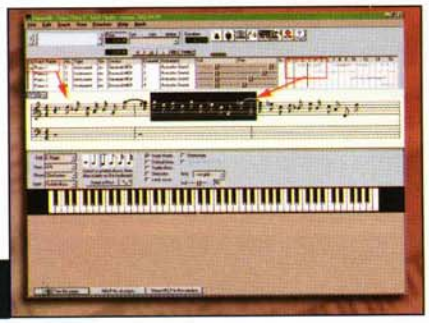
4

Diferentes elementos de las pistas del secuenciador.



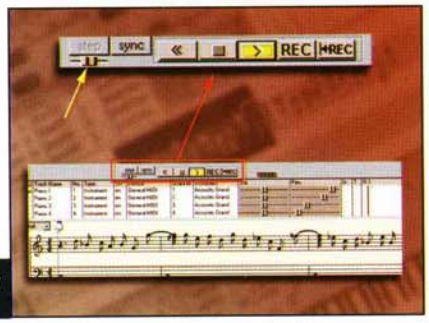
5

A través del pequeño botón del "altavoz" podemos acceder a la configuración de nuestra tarjeta de sonido.



6

En la sección composer es donde podemos editar todos los eventos de cada pista.



7

Algunas opciones de navegación sólo aparecen en la sección composer.

na el canal MIDI de la pista (desde 1 a 16) y en *Instrument* el sonido. Seguidamente, encontramos un deslizador para cambiar el volumen de la pista (*Vol*) y una casilla numérica para definir el panorámico, el cual se modifica a través de un deslizador de color más claro que aparece al hacer clic sobre la casilla *Pan*. Los siguientes parámetros *fx* y *Time Shift* se utilizan para pistas de audio. El primero sirve para asignar un efecto a la pista y el segundo para provocar un retardo en la ejecución de la pista con respecto a las demás. Cada una de las casillas siguientes, representadas por números, corresponden a los compases de la canción y es una representación gráfica del contenido de la pista, la cual se editará en la sección de composición *compose* (Fig. 4).

Para modificar las opciones generales de audio; es decir, la configuración de la tarjeta y del mezclador de Windows, debemos pulsar en el icono . Pulsándolo de nuevo ocultaremos todos los parámetros. Asimismo, podemos modificar el volumen general del sintetizador de la tarjeta con el deslizador junto al icono de configuración "Internal Synth Vol" (Fig.5).

Para crear una nueva pista elegimos la opción *Create* del menú *Track* y a continuación el tipo de pista. Si lo que queremos es borrar el contenido de una pista elegimos la opción *Erase Notes* o *Erase Notes and Properties* del menú *Track*. Para borrar la pista completa elegimos *Delete*. Para hacer una copia completa y exacta de una pista utilizaremos *Clone*. Se creará una pista nueva. También podemos mezclar el contenido de dos pistas con la opción *Merge*. Al elegirla, el programa nos pedirá con qué pista queremos mezclar la seleccionada. Además, nos permitirá mezclar sólo las notas y no los demás eventos seleccionando la opción *Merge Note events only*.

Siguiendo con las opciones del menú *Track*, disponemos de una herramienta para cuantizar toda una pista, se trata de la opción *Quantitize Entire Track*. Además, con la opción *transpose* podemos transportar una sola pista a elegir (*Entire Track*) o toda la canción (*All Tracks in Song*).

PRACTICANDO CON UN EJEMPLO

Vamos a cargar una canción desde disco. Elijamos la opción *Open Song* del menú *File* y buscamos el archivo *FugueGM* que se encuentra en "Extras" del CD adjunto. Se trata de un tema clásico en formato MIDI repartido en 4 pistas. Observamos que en las casillas numeradas de las pistas aparecen una serie de puntitos, son los eventos MIDI de cada compás. Para recorrer los compases debemos utilizar el navegador (gran barra deslizadora de a la izquierda del programa). Para seleccionar una u otra pista podemos utilizar las flechas del cursor. El contenido de cada pista sólo podemos verlo a través de la sección *Compose*, la cual se activa pulsando en el botón "Compose". Seleccionemos la pista número 1 y activemos "Compose". Observamos que aparece un pentagrama con el contenido de música de la pista. Desde aquí, tenemos más opciones para navegar por la pista además del navegador principal. Podemos ir hacia la derecha o izquierda con los cursores y al principio o fin de la pista con "Ctrl" + Cursores. Podemos seleccionar con el ratón cualquier parte del pentagrama o bien con "Shift" + Cursores (Fig. 6).

Para reproducir la canción sólo tenemos que pulsar en el icono y para avanzar paso a paso en la canción disponemos de la opción "step" situado en un pequeño botón a la izquierda del transportador. El pequeño deslizador situado debajo de *step* es para aumentar o disminuir la cantidad de notas de cada paso (Fig. 7).

En el próximo número...

... empezaremos a trabajar con Anvil Studio para crear nuestra primera canción.

Manejo de funciones 3D (II). Animación

En la mayoría de las aplicaciones multimedia y sobre todo en los videojuegos se pueden encontrar geometrías en 3D con algún tipo de movimiento conviviendo con objetos estáticos.

Blitz3D contempla también la posibilidad de controlar modelos con alguna animación, realizada en aplicaciones especiales como CharacterFX. Pero además, también es posible crear nuestra propia animación desde programación. Estudiaremos funciones estándar para la manipulación y creación de animaciones y funciones nuevas añadidas en versiones posteriores a la 1.66.

MANIPULANDO OBJETOS CON ANIMACIÓN

El primer método depende de la secuencia de animación contenida en un modelo 3D y consiste en cargar el *mesh* y con él toda la animación e información de jerarquías (si las tuviese). Para ello, debemos utilizar el comando "LoadAnimMesh":

```
LoadAnimMesh (Nombre_Fichero$
[,entidad_madre] )
```

Esta instrucción cargará cualquier fichero en los formatos .X, .3DS y .B3D. tanto si tuviera animación como si no.

La animación contenida en un objeto 3D está distribuida en secuencias. Quiere esto decir que si, por ejemplo, tenemos el modelo de un ogro, el cual anda, corre y salta, cada una de estas acciones vendrán contenidas en el fichero una detrás de otra, cubriendo un número determinado de fotogramas cada una. Blitz3D al cargar este modelo obtendrá la animación completa, es decir, todos los fotogramas unidos:

```
Modelo = LoadAnimMesh ("Ogro.3ds")
```

Para ejecutar la animación contenida en el modelo utilizaremos la función "Animate":

```
Animate Entidad, [,modo animación]
[,velocidad# animación]
[,secuencia animación]
[,velocidad transición#]
```

Esta función extraerá automáticamente la animación contenida en el modelo y la reproducirá continuamente, desde el primer fotograma hasta el último.

Sólo tenemos que llamarla una sola vez y actualizarla con "UpdateWorld" (Ver "ejemplo1.bb").

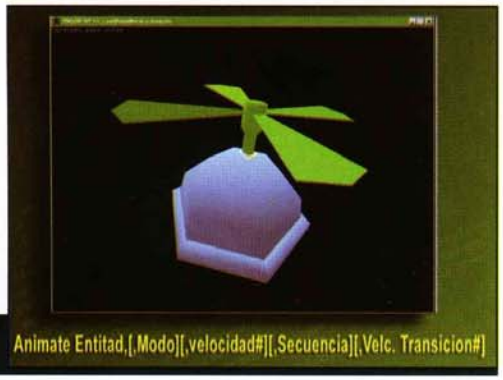
Con escribir: *Animate Modelo* ya funcionará. Sin embargo, posee algunos parámetros opcionales que nos proporcionan el control de la animación de una forma más flexible.

NOTA

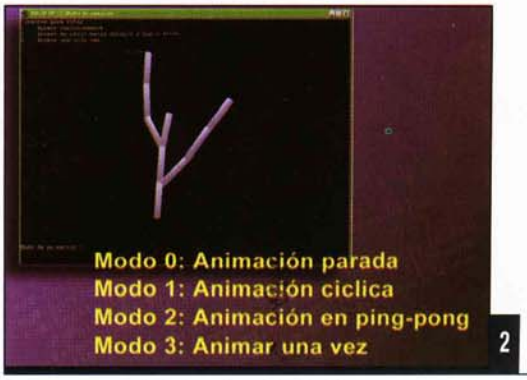
Blitz3D puede manipular objetos animados en los formatos .X (DirectX), .3DS (3D Studio Max), .MD2 (Quake 2) y el reciente formato de Blitz3D para la versión 1.75 y superior, .B3D (muy similar al formato .MD3 y soporta animación por esqueletos, varios mapeados UV y múltiples capas de texturas). (Ver "ejemplo5.bb").

NOTA

La función "UpdateWorld" tiene un parámetro opcional para controlar la velocidad general de todas las animaciones del programa: *UpdateWorld [velocidad_animación#].*



Mediante la función "Animate" podemos reproducir la animación contenida en el modelo cargado con "LoadAnimMesh".



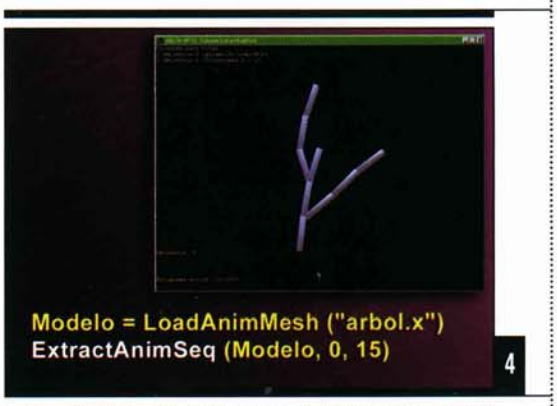
Podemos reproducir la animación de diferentes maneras según el valor del segundo parámetro de la función "Animate".

El parámetro *modo animación* va desde 0 a 3 y define cómo se reproducirá la animación: (Ver "ejemplo2.bb").

- **0.** Animación parada.
- **1.** Animación en ciclo. (Por defecto)
- **2.** Animación en ciclo hacia delante y atrás sucesivamente.
- **3.** Se reproduce la animación una sola vez.

La "velocidad de animación" es un número "float" que determina el "framerate" (fotogramas en un segundo) o velocidad de la animación al reproducirse (Ver "ejemplo3.bb").

El parámetro "Secuencia de animación" especifica qué secuencia de fotogramas contenida en la animación se reproducirá. Por defecto, la primera secuencia adquiere un valor de 0. Este parámetro se utiliza para encadenar animaciones. Ahora bien: inicialmente, un modelo o entidad cargado con "LoadAnimMesh" dispondrá de una sola secuencia de animación. Por lo tanto, aunque nuestro modelo de "ogro" anterior contuviera tres secuencias diferentes, no podríamos distinguirlas desde Blitz3D, ya que se consideran como una sola animación. Para resolver este problema en versiones de Blitz3D anteriores a la 1.75 es necesario tener varios ficheros con una secuencia de animación diferente en cada uno de ellos, por ejemplo, "ogro_andando.3ds", "ogro_corriendo.3ds", etc. Luego, se cargarían uno a uno los ficheros y se encadenarían las animaciones en una misma entidad con la instrucción "LoadAnimSeq":



La función "ExtractAnimSeq" (Versión 1.75) permite adquirir secuencias de animación de un modelo animado evitando tener que tener cada secuencia en ficheros separados.

```
Modelo_ogro = LoadAnimMesh
("ogro_andando.3ds")
LoadAnimSeq Modelo_ogro,
"ogro_corriendo.3ds"
LoadAnimSeq Modelo_ogro,
"ogro_saltando.3ds"
If keydown(205)
  Animate Modelo_ogro,1,.5,1,30
; Ogro  corriendo (secuencia 1)
Else
  Animate Modelo_ogro,1,.5,0,30
; Ogro andando (secuencia 0)
Endif
If keydown(205)  Animate
Modelo_ogro,1,.5,2,30
; Ogro saltando (secuencia 2)
```

En este ejemplo, hemos encadenado a la variable manipuladora "Modelo_ogro" tres animaciones diferentes provenientes de tres ficheros. Hemos construido una animación mayor con tres secuencias, las cuales ya son posibles de diferenciar:

- **Secuencia 0** = Andando
- **Secuencia 1** = Corriendo
- **Secuencia 2** = Saltando

Así que, en la sentencia "Animate Modelo_ogro, 1, .5, 1, 30 ", animamos el modelo del ogro continuamente (primer parámetro) a una velocidad de .5, con la secuencia número 1 (corriendo) El último parámetro (30) se refiere al valor de transición entre una secuencia y otra. Si se coloca un valor 0 de transición, el paso de una secuencia a otra se haría inmediatamente. Mientras que si colocamos un valor mayor (30), se realizará el paso suavemente desde el último fotograma hasta el primero de la nueva secuencia.

Para evitar todo este engorroso proceso de ir sumando secuencias cargadas de ficheros, se introdujo una nueva función a partir de la versión 1.75: "ExtractAnimSeq". Este comando permite "extraer" todas las secuencias contenidas en una animación para poder ser animadas posteriormente con la instrucción "Animate":

```
ExtractAnimSeq (Entidad,
Primer fotograma, Último fotograma
[,secuencia de animación] )
```

También es posible extraer secuencias de fotogramas de una secuencia mayor.

La primera vez que usamos esta función, Blitz3D entenderá que los fotogramas extraídos constituyen la secuencia primera, es decir, la número 1 (la número 0 la constituye la animación completa cargada con "LoadAnimMesh"). La siguiente vez que se utilice, corresponderá a la secuencia segunda (2) Y así sucesivamente:

NOTA

En la función "Animate", un valor negativo en el parámetro de la velocidad provocaría la reproducción hacia atrás de la animación.

Es posible modificar la velocidad de reproducción de la animación por medio del último parámetro de la función "Animate".

Código 1. Uso de ExtractAnimSeq

```
; Secuencia 0. Fotogramas 0 al 90. Ogro andando, corriendo y saltando
Modelo_ogro = LoadAnimMesh ( "ogro.3ds")
ExtractAnimSeq (Modelo_ogro, 0, 20)     ; Secuencia 1. Fotogramas 0 al 20.   Ogro andando
ExtractAnimSeq (Modelo_ogro, 21, 50)    ; Secuencia 2. Fotogramas 21 al 50. Ogro corriendo
ExtractAnimSeq (Modelo_ogro, 51, 90)    ; Secuencia 3. Fotogramas 51 al 90. Ogro saltando
Animate Modelo_ogro, 1, .5, 1           ; Animando continuamente al ogro andando  con velocidad .5
```

(Ver "ejemplo4.bb" o "ejemplo4.exe").

OBTENIENDO INFORMACIÓN

Para tener un control más exhaustivo de los procesos de animación, disponemos de varias instrucciones que nos ayudarán a obtener información muy valiosa.

En ocasiones, necesitaremos saber si una animación sigue su curso o ha terminado. Para saberlo, disponemos del comando "Animating", el cual retorna "True" si la animación de una entidad no ha acabado: Animating (Entidad).

```
If Animating (Modelo_ogro) =
False  Text 10,10,
"La animación ha terminado"
```

NOTA

Cuando se carga un modelo .MD2 sólo lo hace la geometría, las texturas no. Por ello, es necesario cargarlas y asignarlas a parte.

Podemos también saber cuántos fotogramas tiene la animación de un fichero cargado mediante la instrucción "AnimLenght". Esta función devuelve un número entero correspondiente a la longitud de la animación de una entidad:

```
Text 10,10,"El número de
fotogramas es: "+AnimLenght
(Modelo_ogro)
```

Para saber en qué fotograma se encuentra la animación en curso podemos utilizar "AnimTime". Esta función devuelve un valor flotante correspondiente al fotograma en curso:

```
AnimTime#  ( Entidad )
```

Por último, disponemos de otro comando que nos devolverá el número de la secuencia actualmente en reproducción: *AnimSeq (Entidad).*

ANIMACIÓN EN FORMATO .MD2

Además de los formatos explicados anteriormente, Blitz3D permite la manipulación de modelos con animación en formato .MD2. Los comandos normales de animación no funcionan para este tipo de modelo ya que el formato .MD2 tiene su propio grupo de comandos de animación.

Para cargar un modelo .MD2 disponemos del comando:

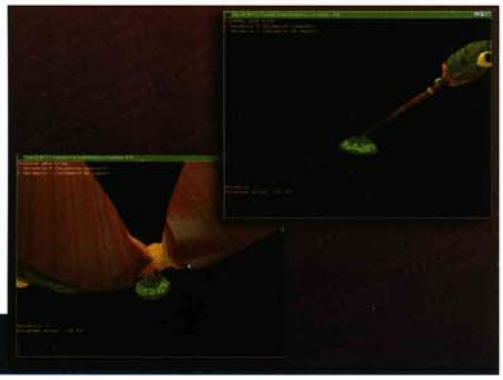
```
Modelo = LoadMD2 ( Fichero MD2
[, Entidad madre] )
```

Al cargar el modelo toda la animación e información de jerarquías se cargará con él. Las secuencias se almacenan de forma secuencial, así que sólo tenemos que saber en qué fotograma empieza y termina cada secuencia. Por este motivo la función para animar el modelo cambia con respecto a la función estándar "Animate". Disponemos entonces de "AnimateMD2", que anima la secuencia del modelo contenida entre fotogramas.

```
AnimateMD2 modelo MD2 [,modo
animación] [,velocidad#]
[,primer fotograma] [,último
fotograma] [,transición#]
```

Los parámetros funcionan exactamente igual que en las funciones de animación estándar.

```
Modelo_ogro = LoadMD2 ("ogro.md2")
Textura_ogro = LoadTexture
("textura_ogro.bmp")
EntityTexture Modelo_ogro,
Textura_ogro
```



5

El nuevo formato ".B3D", soportado a partir de la versión V. 1.75, permite la animación por esqueletos, varios mapeados UV y múltiples capas de textura.

6

Blitz3D permite el manejo de modelos en formato .MD2. Utiliza un sistema diferente de animación, por lo que las funciones para utilizarlo son distintas de las normales.

```
AnimateMD2 Modelo_ogro,1,.5,0,20
; Ogro andando continuamente
```

Disponemos además de comandos propios para obtener información acerca de las animaciones en curso. Para obtener el momento exacto donde se encuentra la animación, disponemos de la función:

```
MD2AnimTime# ( Modelo MD2 )
```

Por ejemplo, si la animación se encuentra en la mitad del fotograma 10 y 11 obtendremos un valor de 10.5. Para saber los fotogramas totales de una animación de un modelo MD2 utilizaremos:

```
MD2AnimLength ( Modelo MD2 )
```

Y por último, para saber si una animación ha terminado o no:

```
MD2Animating ( Modelo MD2 )
```

(Ver "ejemplo6.bb").

CREANDO NUESTRAS PROPIAS ANIMACIONES DESDE PROGRAMACIÓN

También es posible generar animaciones con primitivas u otros objetos 3D con el uso de la programación. Vamos a trabajar como lo hace una aplicación para generar animaciones como, por ejemplo, CharacterFX, utilizando "Keyframes". El procedimiento es bien sencillo.

Supongamos que tenemos un cubo en una posición de la pantalla y queremos crear una animación, consistente en 30 fotogramas, en los que el cubo se desplaza hacia otra posición. Pues bien, situamos el cubo en posición:

```
Cubo = CreateCube()
PositionEntity Cubo, -8, 0, 5
SetAnimKey Cubo, 0
```

Luego creamos un keyframe en la posición original (0) con "SetAnimKey".

Desplazamos el cubo y grabamos otro keyframe en el fotograma 30 en la nueva posición:

```
PositionEntity Cubo,10,0,5
SetAnimKey  Cubo, 30
```

Blitz3D creará los fotogramas intermedios automáticamente basándose en los valores de posición, rotación y traslación de los objetos en los puntos definidos por los "keyframes".

Una vez creados los puntos de la animación, la creamos con *AddAnimSeq Entidad, longitud en fotogramas:*

```
AddAnimSeq Cubo, 30
```

Cada vez que apliquemos esta función se creará una secuencia nueva, siempre en orden secuencial. Por lo tanto, si después de desplazar el cubo queremos rotarlo y desplazarlo a la vez hacia otro lugar, tendremos que crear otro keyframe y sumar la animación:

```
PositionEntity Cubo,0,0,10
RotateEntity Cubo,0,90,0
SetAnimKey Cubo,60
AddAnimSeq Cubo, 60
```

En este caso tendremos dos secuencias de animación, una desde el fotograma 0 al 30 (secuencia número 0) y otra desde el 31 al 60 (secuencia número 1). Para animar, sólo tenemos que utilizar la función "Animate" como se ha descrito anteriormente.

A continuación se muestran con detalle todos los parámetros de "SetAnimKey":

```
SetAnimKey Entidad, fotograma
[, activar keyframe de posición]
[,activar keyframe de rotación]
[,activar keyframe de tamaño]
```

Los parámetros opcionales de activación están por defecto a 1 ("True") y sirven para incluir en el keyframe los cambios de posición, rotación y tamaño del objeto. Si en el ejemplo anterior, en el segundo "Keyframes" pusiéramos:

```
PositionEntity Cubo, 10, 0, 5
SetAnimKey Cubo, 30, False,
True, True
```

el cubo no se desplazaría en absoluto hacia la nueva posición.

Esta manera de crear animaciones es muy interesante sobre todo para poder construir nuestras propias aplicaciones de grabación de movimiento en Blitz3D. (Ver "ejemplo7.bb").

```
PositionEntity Modelo, -6, -2, 5
; Creamos un keyframe
SetAnimKey Modelo, 20
; Sumamos las secuencias para formar la animación
AddAnimSeq Modelo,20
Animate Modelo,1,Velocidad_animacion
```

7

La creación de animación en tiempo de ejecución por sistema de keyframe también es posible en Blitz3D.

NOTA

Para crear una animación desde programación es necesario seguir los siguientes pasos:
1. Posición inicial modelo + "SetAnimKey modelo,fotogramas-A". Primera secuencia.
2. Siguiente posición modelo + "SetAnimKey modelo, siguientes fotogramas-B". Segunda secuencia.
3. Sumar secuencias para crear animación con "AddAnimSeq modelo, fotogramas-A+fotogramas-B.

En el próximo número...

... estudiaremos las funciones para el manejo de cámaras.

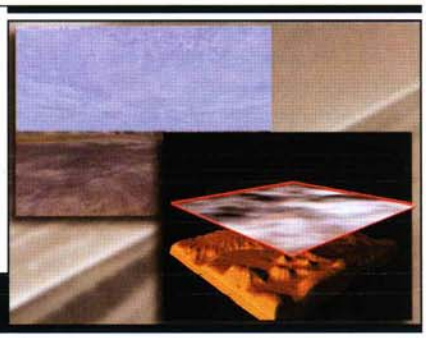
Creación de cielos para un entorno

Hay muchos métodos para la creación de cielos para un entorno. Probablemente, el más fácil sea mediante el uso de un plano situado a una altura determinada del terreno. Este método ha sido utilizado en nuestro juego "Zone of Fighters".

Otro método sería utilizar una esfera o un cubo con textura que cubra todo el entorno. Y por último, el método más sofisticado y quizás el más utilizado hoy día en videojuegos sea mediante el mapeado cúbico.

UTILIZANDO PLANOS

Utilizar un plano texturizado es un método muy sencillo, eficaz y fácil de implementar en un juego. Generalmente, se suele usar en niveles de interior en donde sólo se ve el cielo a través de las ventanas o huecos en el techo. También se puede utilizar en entornos de exterior en donde se impide ver el horizonte por el uso de niebla. Además, es muy común el uso de scroll de textura para generar la sensación de movimiento de nubes. En la figura 1 se muestra un esquema gráfico de cómo se utiliza (Ver ejemplo "cielo_plano.bb").



1

Esquema y resultado de un cielo creado por el sistema del plano.

El procedimiento básico es crear un *mesh* plano, texturizarlo y situarlo sobre el terreno o nivel (BSP, mesh, etc.):

```
Cielo=CreatePlane()
TextureEntity Cielo,Textura_cielo
FlipMesh Cielo
PositionEntity Cielo,0,1000,0
```

En este caso, "FlipMesh Cielo" funciona igual que "RotateEntity Cielo,180,0,0". Lo que hacemos es dar la vuelta al plano para que la textura mire hacia abajo.

Seguidamente, podemos desplazar la textura para simular el movimiento del cielo.

UTILIZANDO ESFERAS O CUBOS

La utilización de una esfera produce un resultado realmente interesante, ya que podemos mover el cielo girando el *mesh* o desplazando la textura. En el uso de esferas hay que tener en cuenta los vértices de unión de los segmentos, sobre todo por la deformación que la textura sufre en los polos de la esfera. Básicamente, consiste en posicionar el terreno en el interior de una gran esfera texturizada. Al estar en el interior no veremos la textura a no ser que le apliquemos la función "FlipMesh" a la primitiva (Ver ejemplo "cielo_esfera.bb"):

```
Cielo=CreateSphere()
ScaleMesh Cielo,
escala_X,escala_Y,escala_Z
```

2

Esquema y resultado de un cielo creado por el sistema de esfera.

```
TextureEntity Cielo,Textura_cielo
FlipMesh Cielo
PositionEntity Cielo,0,1000,0
```

Por otro lado, la utilización de un cubo funciona exactamente igual, a diferencia, claro está, del resultado visual. (Ver ejemplo "cielo_cubo.bb").

MAPEADO CÚBICO

Este método consiste en crear un cubo polígono a polígono y texturizar cada una de las caras con una textura diferente. Es el más eficaz y utilizado, ya que el uso de primitivas como esferas o cubos tiene un par de inconvenientes. En primer lugar, no podemos utilizar varias texturas

3

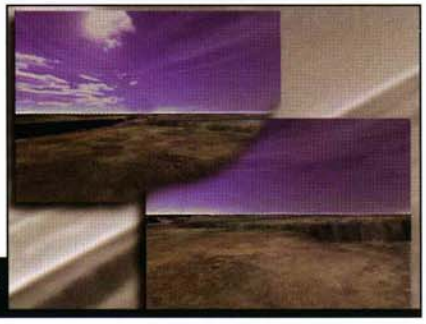
Esquema y resultado de un cielo creado por el sistema del cubo.

4

Textura desglosada para un mapeado cúbico.

para generar un cielo completo; es decir, mostrando el sol y nubes distintas según se mire. Si utilizamos una esfera, la textura se adaptará a ella cubriéndola por completo y si usamos un cubo, se aplicará automáticamente en cada una de las caras. Por lo tanto, no tenemos ningún tipo de control al respecto.

En segundo lugar, al utilizar "FlipMesh" en un cubo texturizado las uniones de las caras no quedan perfectas y es posible apreciar, cuando la cámara se acerca, las aristas de color negro. Así que el mejor método resulta ser la aplicación de un mapeado por caras a un cubo modelado desde programación. Es el más complejo de todos pero los resultados son realmente buenos.



5

Resultado del sistema de mapeado cúbico para implementar un cielo.

PREPARANDO LAS TEXTURAS

Para este tipo de mapeado debemos preparar cinco texturas diferentes, una por cada vista del cielo: lateral izquierdo, lateral derecho, vista frontal, vista trasera y vista superior. Evidentemente, para la vista inferior no es necesaria ninguna textura, ya que nunca se vería. En la figura 4 se muestra un esquema gráfico de cómo va colocada cada una de ellas.

CREANDO Y TEXTURIZANDO EL CIELO

El siguiente proceso será modelar cada una de las caras del cubo por medio de triángulos. Éstos, a su vez, serán construidos a partir de la unión de vértices. En definitiva, crearemos una superficie por cara, con dos triángulos y cuatro vértices, y todas las superficies formarán un *mesh*. Para llevar a cabo el texturizado, es necesario la utilización de *brushes* para pintar cada superficie:

```
Cielo=CreateMesh()
;CARA FRONTAL
Textura=LoadBrush("Textura_Frontal.png")
Lado=CreateSurface(Cielo, Textura)
AddVertex Lado,-1,1,-1,0,0
AddVertex Lado,1,1,-1,1,0
AddVertex Lado,1,-1,-1,1,1
AddVertex Lado,-1,-1,-1,0,1
AddTriangle Lado,0,1,2
AddTriangle Lado,0,2,3
FreeBrush Textura
;CARA TRASERA
Textura=LoadBrush("Textura_trasera.png")
Lado=CreateSurface( Cielo,Textura)
AddVertex Lado,1,1,1,0,0
AddVertex Lado,-1,1,1,1,0
AddVertex Lado,-1,-1,1,1,1
AddVertex Lado,1,-1,1,0,1
AddTriangle Lado,0,1,2
AddTriangle Lado,0,2,3
FreeBrush Textura
...
...
```

El resto de las caras se creará de la misma manera. El procedimiento por cada lado es el siguiente:

- 1. Se crea una superficie, la cual pintamos con el *brush* "Textura".
- 2. Se crean cuatro vértices en esa superficie.
- 3. Se unen los vértices de tres en tres para formar triángulos.
- 4. Se libera de la memoria el *brush* cargado.

Para comprender mejor este procedimiento es preciso observar el esquema de la figura 5.

- 5. Una vez creado el *mesh*, lo escalamos, posicionamos y damos la vuelta.

```
ScaleMesh Cielo,5000,5000,5000
PositionMesh Cielo,0,0,0
FlipMesh Cielo
EntityFX Cielo,1
Return Cielo
```

En el ejemplo contenido en el CD "mapeado_cubico.bb" se utiliza este sistema para rodear un terreno con un cielo y la técnica del plano para crear nubes en movimiento.

En el próximo número...

... aprenderemos a utilizar imágenes predefinidas para crear nuestra propia fuente de letras.

Juegos de estrategia (III). Estrategia en tiempo real

Continuando con la serie de juegos de este género tan prolífico, dedicaremos este número a aquellos títulos que no siguieron una tendencia belicista o destructiva para desarrollar su argumento, sino que tomaron como base la construcción de ciudades, fábricas, empresas o cualquier gestión de recursos a gran escala.

Pero antes, no queremos olvidar un subgénero bélico que ha levantado pasiones en multitud de seguidores de la estrategia, nos referimos a los RPS.

ROLE PLAYER STRATEGY (RPS)

Nacido de varios conceptos de estrategia, este tipo de juego contribuye a proporcionar a la historia de los videojuegos grandes títulos, nos referimos a los RPS. Básicamente consiste en que el jugador maneja sólo un puñado de unidades, cada una capaz de realizar una tarea determinada. Quizás el título más conocido a nivel mundial sea *Comandos: Behind Enemy Lines* (1998) y su segunda y tercera *parte Commandos 2: Men of Courage* (2001) y *Commandos 3* de la desarrolladora española Pyro Studios. La saga constituye el plato fuerte de Pyro en este género y destaca, sin duda, por la exquisita calidad gráfica de los personajes y escenarios detallados al milímetro. Este concepto de tanto éxito fue adquirido rápidamente por muchas desarrolladoras y empezaron a aparecer títulos de mayor o menor calidad gráfica pero que simulaban el mismo tipo de estrategia. Por ejemplo, en *Veiled Threat* (Lupine Games, 2000) manejamos un grupo de mercenarios en tiempos de la Segunda Guerra Mundial. También en el mismo ambiente, pero en un entorno 3D encontramos *Hidden & Dangerous* (Take2 Interactive, 2000). Sin embargo, hay dos títulos que calcan el estilo gráfico impuesto por *Comandos*, nos referimos a *Desperados: Wanted Dead or Alive* (Infogrames, 2001) o *Robin Hood: The Legend of Sherwood* (Strategy First, 2002).

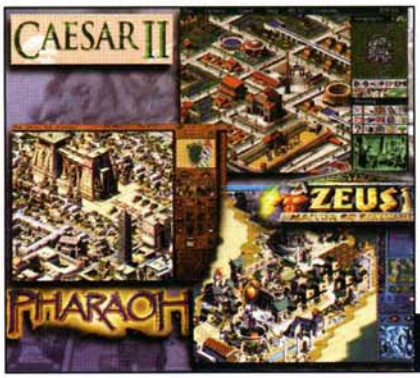
ESTRATEGIA CONSTRUCTIVA

Al igual que ocurrió en la estrategia por turnos, aparecieron también tendencias de género estratégico no belicista en el ámbito de la creación y gestión de todo tipo de ciudades a lo largo de todas las épocas. Este argumento casi no ha cambiado desde el primer constructor en tiempo real. Los conceptos se mantienen a diferencia, evidentemente, de mejoras gráficas. Debemos remontarnos al año 1989 cuando la casa Maxis saca al mercado *Sim City*. En este juego debemos construir y gestionar una ciudad entera. Un argumento aparentemente simple pero de una complejidad incuestionable que generó una gran avalancha de títulos. Maxis siguió publicando nuevas versiones de *Sim City: Sim City 2000* (1993), *Sim City 3000* (1998) y el actual *Sim City 4* (2002)

A su lado se unen otras desarrolladoras que optan por el mismo argumento como la serie *Caesar* (Sierra, *Caesar I* 1994, *Caesar II* 1995 y *Caesar III* 1998) inspirada en el imperio romano o *Pharaoh* (Sierra, 1999) en el antiguo Egipto. También apareció *Zeus: Master of Olympus* (Sierra, 2001) situada en la antigua Grecia.

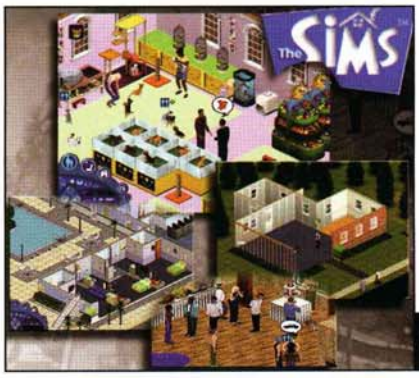
1

La serie Commandos y Hidden & Dangerous, primeros RPS.



2

Ejemplo de estrategia constructiva de ciudades ambientadas en distinta épocas de la Historia.



3

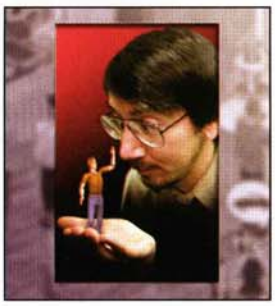
La simulación social en The Sims adquiere matices de estrategia cuando tienes que controlar todo un vecindario.

LA BIOGRAFÍA...

WILL WRIGHT

Creador del **Shooter 3D**

Junto con Adrian Carmack y John Romero fundó ID Software, compañía creadora del shooter en 3D. Está considerado como el programador de juegos y tecnología 3D en tiempo real más importante e innovador del mundo. Su filosofía es ir siempre de la mano con los nuevos avances tecnológicos en gráficos para ordenador. Gracias a esto se ha convertido en un gurú de los videojuegos con creaciones como la saga *Wolfenstein, Doom* y *Quake*. Es amante de los viejos juegos *arcade* como *Pacman* o *Space Invaders* y aprendió solo a programar. Vendió su primer juego cuando estaba en el instituto, un RPG llamado *Shadowforge*. En 1990 entró a trabajar en SoftDisk, donde programó la serie *Commander Keen*. Su tecnología ha sido muy solicitada y el motor del *Quake II* sirvió para desarrollar juegos como *Soldier of Fortune* o *Half Life*. Carmack es amante de la enseñanza y son conocidas sus andaduras por el mundo dando infinidad de conferencias. Una muestra de esta afición es la posibilidad de adquirir por Internet el código de sus juegos más importantes.



Will Wright

La idea de crear y gestionar tu propia ciudad se extendió a otros ámbitos siguiendo la misma temática. Aparecen entonces diferentes juegos que adaptan este concepto a otras situaciones como la gestión completa de un parque de atracciones en *Theme Park* (Bullfrog, 1994) o la gestión ferroviaria que Microprose ofrece en *Railroad Tycoon* (1995), *Roller Coaster Tycoon* (1998), o la segunda parte *Railroad Tycoon 2* (Pop Top Ware, 1998) pero esta vez con gráficos en 3D. Sin embargo, aquí no queda todo. Numerosos títulos salen a la luz con multitud y disparatados argumentos. Comienza de nuevo Maxis con constructor de hormigueros *Sim Ant* (1995) y continúa de nuevo Bullfrog con su juego *Theme Hospital* (1997) donde tu misión será gestionar lo mejor posible todas las partes de un hospital. Ese mismo año Interplay publica *Evolution (*Crossover Technologies*)* en el que tienes que gestionar la evolución de las especies hasta la época de los dinosaurios. Cambiando de tercio, la desarrolladora Mucky Foot nos traslada al espacio con su trabajo *Startopia* (2001) que con unos gráficos divertidos y originales nos invita a gestionar una estación espacial repleta de extraterrestres. La buena acogida de este género, en parte debido a que son programas que pueden ser jugados por toda la familia, provoca una gran producción que conlleva a una mayor diversidad de temas. Es el caso de *Pizza Syndicate* (Software 2000, 1999), una de las más simpáticas alternativas, en el que debemos construir un imperio comercial en torno al negocio de las pizzerías. En esa misma tendencia en el sector económico, Pop Top Ware publica *Tropico* (2001), en el que tienes que controlar toda una isla.

Todo este ir y venir de juegos de estrategia económica y social provoca un seguimiento por parte de expertos de diferentes géneros interesados en utilizar este tipo de entretenimiento como motor de enseñanza en niveles profesionales reales. Es el caso del simulador de bolsa *Wall Street Trader* (Monte Cristo / Electronic Arts, 2001), considerado el mejor en su categoría.

Pero quizás el mayor fenómeno social acaecido desde la invención de los juegos de estrategia de carácter no belicista haya sido el juego *The Sims* (Maxis, 2000). En él, debemos crear y controlar a una familia. Además, podemos construir todo un vecindario y organizar todas las relaciones entre sus habitantes. Está considerado el mejor juego de tipo social y posee una IA increíble en la que se barajan casi todas las premisas que existen en las relaciones humanas. Desde su publicación, sigue estando en los primeros puestos de las listas de ventas y su mayor éxito radica en su sencilla interfaz de uso y la posibilidad de juego para toda la familia sin excepción. Actualmente, existe más de media docena de expansiones todas orientadas a ofrecer más de lo mismo y a situar a nuestras familias virtuales en diferentes lugares y situaciones: *Sim House Party, Vacation, Sims más vivos que nunca*, etc. Gráficamente todos están cortados con la misma tijera, utilizan una representación en 2D con perspectiva isométrica y con una estética bastante estilizada.

Todavía nos queda el estudio de un subgénero de estrategia donde el principal argumento es otorgar al jugador el poder de modificar la trayectoria del juego a voluntad. Se trata de los simuladores de "Dios". Sin embargo, dejaremos su estudio para el próximo número en el que también hablaremos sobre el tipo de estrategia más conocida por todos, los puzzles.

En el próximo número...

... terminaremos nuestro repaso a la historia de los juegos de estrategia en tiempo real.

Cuestionario Videojuegos 12

Preguntas

1. En Blitz3D, ¿para qué tipo de operaciones se utiliza la instrucción "LoadAnimMesh"?
2. Hemos creado el modelo de un hombre, el cual posee el movimiento de andar. Escribe el código para observar esa animación en Blitz3D.
3. Define una estructura simple y una función para crear disparos sólo para controlar su desplazamiento.
4. ¿Cómo podemos actualizar todos los disparos del juego en el bucle de control del juego?
5. ¿Cómo podemos determinar un punto de unión como final de jerarquía en Character FX?
6. ¿Cómo podemos variar la velocidad de reproducción de la animación en Character FX?
7. ¿Qué tipo de pistas podemos crear en Anvil Studio?
8. ¿En qué sección de Anvil Studio editamos los eventos MIDI de una pista?
9. Enumera al menos tres métodos para crear un cielo en Blitz3D.
10. Enumera las fases para programar un cielo utilizando mapeado cúbico.

Respuestas al cuestionario 11

▷ **1.** Para realizar esta operación es necesario activar el búfer de texturas con "SetBuffer TextureBuffer()" para que al dibujar los haga directamente sobre el objeto.

▷ **2.** Los *brushes* o pinceles son un grupo de propiedades del material que se le aplican a una entidad. Para crear un *brush* nuevo se utiliza la instrucción "CreateBrush" y para crearlo a partir de un fichero de imagen "LoadBrush".

▷ **3.** Para lograr una aceleración se puede utilizar una variable de desplazamiento y otra de aceleración. Además, es necesario controlar los topes máximos y mínimos en la velocidad:

```
If MouseDown(boton2)  And velocidad<velocidad_max
    velocidad# = velocidad + aceleracion#
    MoveEntity bionave, 0, 0, velocidad
  Else
    Velocidad = velocidad - (aceleracion#*2)
    MoveEntity bionave, 0, 0, velocidad
EndIf
If velocidad<4 velocidad=4
```

▷ **4.** Mediante la instrucción "PointEntity": *PointEntity camara,objetivo.*

▷ **5.** Modificando la opción *Hue* situada en la sección *Base* en la pestaña *Advanced Behaviour.*

▷ **6.** Aplicando la opción *Reverse Vertex Order* del menú *Face.*

▷ **7.** Disponemos de dos tipos de canales: canales de muestras "sampler" y de sonidos sintetizados "TS404".

▷ **8.** Primero, seleccionamos la pista o canal. Abrimos la ventana *Song settings* pulsando en *FX1* de la ventana *Channel settings.* Seleccionamos un efecto de la lista *Favorites* y pulsamos en el icono del enchufe para activar el efecto.

▷ **9.** Mediante la instrucción "BSPAmbientLight".

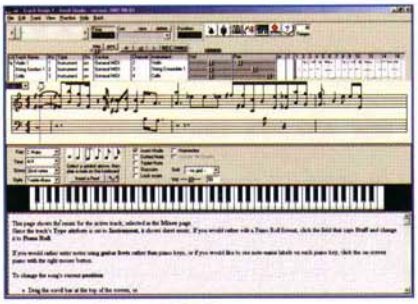
▷ **10.**

```
Linterna=CreateLight(3,camara)
LightConeAngles Linterna,0,60
```

Contenido CD-ROM 12

▶ AUDIO

■ Anvil Studio
Nueva oportunidad para hacernos con este estupendo programa con el que podrás componer tus propias melodías.



■ FractMus 2000
Con esta curiosa aplicación podrás crear música a partir de fórmulas matemáticas.

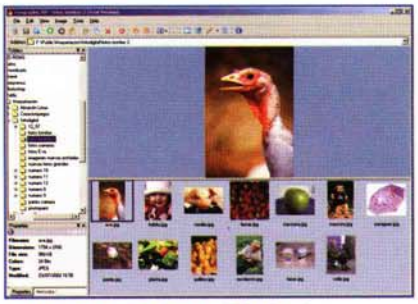
■ Mellosoftron 3.4
Convierte tu ordenador en un editor de música en vivo y crea tus propios samples.

■ Orion Virtual Studio Pro 2.3
Con este programa podremos realizar remixes como en un estudio profesional.

▶ DISEÑO 2D

■ Drawlt 3.1
Práctica herramienta de dibujo vectorial y de ilustración muy sencilla de usar.

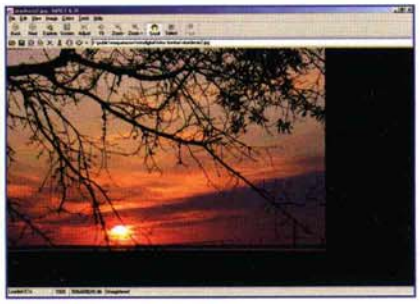
■ Firegraphic XP 5.5.4
Edita imágenes e imprímelas usando filtros y muchas otras características.



■ Nature Painter Digital Canvas 1.1
Aprende distintas técnicas pictóricas con la ayuda de este programa.

■ PhotoWizard
Excelente herramienta para añadir sorprendentes efectos a las fotografías en muy poco tiempo.

■ AI Picture Utility 6.3
Visor de imágenes que contiene además múltiples utilidades.



■ PicViewer 2.7.4
Otro visor de imágenes en muchos formatos.

▶ DISEÑO 3D

■ AC3D 3.5
Modela y crea escenas en tres dimensiones con este programa.

■ Image Styles 3.4
Complemento ideal para la creación de los paisajes del juego en muy poco tiempo.

■ Mimmix FE 2.0
Crea simpáticos personajes en tres dimensiones con esta divertida aplicación.

■ Protix 3D Builder 1.3
Creación de imágenes en 3D con posibilidad de grabarlas en formato DirectX.

■ Character FX
Nueva oportunidad de conseguir esta excelente aplicación.

▶ PROGRAMACIÓN

■ Defect Tracker 2.0
Organiza el desarrollo del juego con esta utilidad que te permitirá reportar errores.

■ Fast Bug Track 3.1
Software de mantenimiento de cambios en los proyectos de programación.

■ Gran PM 1.6.5
Control de los proyectos y de sus errores.

■ PlayerPro SDK 5.9.4
Kit de desarrollo para poder incorporar sonido en los programas en C y C++.

▶ JUEGOS

■ Commandos: Behind Enemy Lines
Popular juego de estrategia con el que pasarás muy buenos momentos.

■ SimCity 3000
Demo del excelente juego que aún bate récords de ventas en el que deberás construir y manejar una ciudad entera.

■ Startopia
Original juego que nos invita a gestionar una estación espacial repleta de extraterrestres.

■ Zone of fighters
Nuestro juego, incluido como siempre.

▶ VÍDEO

■ Elektronika Live 1.0
Excelente aplicación para crear auténticos shows visuales en directo.

■ JPGAvi 1.0
Crea estupendos vídeos en formato AVI a partir de imágenes en JPG.

■ Video Edit Magic 1.5
Captura vídeo y edítalo cómodamente desde tu ordenador.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.

Actuando con el entorno

Una vez completado el estudio de cómo controlar nuestra bionave de combate por el jugador, es preciso implementar un sistema de colisiones que proporcione una total interacción con el entorno.

Quizás ésta sea la parte más importante en el desarrollo de un juego, ya que puede determinar la credibilidad del mundo que queremos plasmar en el monitor. Así que vamos a intensificar nuestros esfuerzos en la detección, por parte de nuestra nave y de los disparos, del terreno, árboles, rocas y edificios, así como de los animales, plantas y otros enemigos.

¿CÓMO DESPLAZAR LA BIONAVE POR EL TERRENO?

No hace falta recalcar la importancia del terreno en un juego como "Zone of Fighters". En él se desarrolla toda la acción y es el soporte de todo el entorno. Es obvio que su presencia en el mapa de colisiones del juego es primordial.

Como explicamos en la sección de Blitz3D del número 8, lo que primero debemos hacer es crear un mapa de colisiones formado por las entidades involucradas. Para ello, es preciso asignar a cada una de ellas el tipo de colisión con el comando "EntityType". En realidad, se trata de establecer a cada entidad el número de identificación que tendrá dentro del mapa. Por ejemplo, podemos definir el terreno como entidad 1 o la bionave como entidad 4 y así con todas las demás:

```
EntityType terreno1, 1
EntityType camara, 2
EntityType bionave, 4
....
```

Realmente resulta sencillo. Pero supongamos que queremos definir un gran número de tipos de entidades. Llegará un momento en el que será difícil distinguir a qué tipo pertenece una entidad cualquiera cuando tengamos que hacer referencia a ellas en nuestro código. Para solucionar el problema, asignamos a cada número un nombre por medio de constantes. Así nuestra definición anterior quedaría:

```
Const ENTIDAD_TERRENO = 1
Const ENTIDAD_CAMARA = 2
Const ENTIDAD_BIONAVE = 4
EntityType terreno1, ENTIDAD_TERRENO
EntityType terreno1, ENTIDAD_CAMARA
EntityType bionave, ENTIDAD_BIONAVE
....
```

Como se puede comprobar, de esta manera es mucho más sencillo saber a qué tipo nos estamos refiriendo. Por lo tanto, en el módulo "Definiciones.bb" definiremos estos tipos de entidades mediante constantes:

```
Const ENTIDAD_TERRENO =  1
Const ENTIDAD_CAMARA  =  2
Const ENTIDAD_BIONAVE =  4
```

Una vez definidos los tipos debemos asignarlos para el mapa de colisiones (recordemos que la cámara es una entidad vinculada a la bionave, así que se comportará de la misma manera):

```
EntityType terreno1, ENTIDAD_TERRENO
EntityType terreno1, ENTIDAD_CAMARA
EntityType bionave, ENTIDAD_BIONAVE
```

NOTA

Todos los procesos de colisión que afecten a la bionave influirán de igual manera a la cámara como entidad ligada a ella.

La cámara se comporta exactamente igual que la bionave en el proceso de colisión con el terreno.

Para realizar todas estas definiciones creamos la función "Mapa_Colisiones", la cual situamos en el módulo de gestión del juego "Fjuego.bb" (Ver Fig.1).

Ya estamos preparados para activar la colisión entre ambas entidades mediante la instrucción "Collisions":

```
Collisions ENTIDAD_CAMARA,
ENTIDAD_TERRENO, 2, 3
Collisions ENTIDAD_BIONAVE,
ENTIDAD_TERRENO, 2, 3
```

Hemos utilizado como método el sistema de esfera a polígonos para que la bionave detecte el terreno. Es el más recomendable para utilizar con terrenos, ya que sólo tenemos que pasar al sistema de colisiones de Blitz3D el tamaño de la primera entidad. También hemos utilizado como respuesta al choque entre las dos entidades el sistema de deslizamiento. De esta forma, conseguimos que la bionave, en vez de quedarse parada al chocar con una elevación, se deslice suavemente. Ya sólo nos queda definir la esfera invisible que rodea a la bionave y que sirve de referencia al sistema de colisiones para poder realizar las detecciones con los polígonos del terreno.

Utilizando "EntityRadius" aumentamos el rango de acción del sistema de colisión "esfera - a - polígonos".

Utilizaremos el comando "EntityRadius" que aplicamos en el momento de cargar el modelo de la bionave en la función "crear_modelos" del módulo "funcpantaudio.bb" (Ver Fig.2):

```
bionave=LoadMesh ("c:\zone of
fighters\modelos\bionave.3ds")
pivote_bionave=CreatePivot(bionave)
EntityTexture bionave,textura_bionave
EntityRadius bionave,55
...
```

Con todos estos procedimientos ya hemos logrado que la bionave no atraviese el terreno y que se deslice siguiendo sus irregularidades (Ver Fig.3).

DETECTANDO EL DECORADO

Ahora mismo, si lanzamos a la bionave contra un edificio, un árbol o una roca, los atravesaría sin más. Esto es debido a que todavía no hemos definido su presencia en el mapa de colisiones del juego. Realmente, el paso para hacer esto es prácticamente igual que el realizado para la detección del terreno. En primer lugar definimos las constantes para los tipos de colisión del decorado. Así que podemos definir cada elemento anterior como sigue:

```
Const ENTIDAD_ALMACEN   = 5
Const ENTIDAD_ALMACEN2  = 6
Const ENTIDAD_PUENTE    = 7
Const ENTIDAD_GENERADOR = 8
Const ENTIDAD_ROCA1     = 9
Const ENTIDAD_ROCA2     = 10
Const ENTIDAD_ARBOL1    = 11
Const ENTIDAD_ARBOL2    = 12
...
```

Si observamos bien, estas definiciones no están verdaderamente optimizadas. Si nos paramos a pensar, veremos que lo que buscamos con la definición de colisiones es una respuesta determinada del sistema al choque de una entidad con otra. Según el diseño del juego sabemos que hay entidades del decorado que sufrirán algún tipo de desplazamiento cuando son alcanzadas por un disparo y otras que permanecerán intactas. Sabiendo este dato podemos deducir que en el juego existen dos tipos de entidades de decorado claramente diferenciadas, aunque para ambos tipos, la bionave se comportará de la misma forma en el momento de colisionar con ellos. Es evidente, también, que si la bionave colisiona con la entidad "Roca1" o "Almacen1" los efectos serán los mismos que si lo hiciera con la entidad "Roca2" o "Almacen2". Así que lo más lógico sería englobar a estas entidades en el mismo tipo de colisión denominándolas, por ejemplo, como del tipo "EDIFICIO". Así que podemos resumir la anterior definición de constantes de la siguiente manera:

```
Const ENTIDAD_EDIFICIO = 5
Const ENTIDAD_ARBOL    = 6
```

Todas las construcciones: almacenes, generadores, puentes y rocas, pertenecerán al grupo de los "edificios", el cual constituye el decorado estático. Y todos los tipos de árboles pertenecen al grupo de elementos del decorado móviles.

Por lo tanto, y siguiendo esta premisa, podemos reducir enormemente las definiciones y por consiguiente el mapa de colisiones para el resto de entidades. De esta manera obtenemos un mapa más limpio, pequeño y eficaz, lo cual favorece a obtener un mayor rendimiento en el juego. Ya sólo nos queda asignar estos tipos al mapa de colisiones y activarlos. Las asignaciones de los elementos estáticos y movibles del decorado la haremos en el momento de crear cada una de estas entidades. Así que trasladamos esta cuestión al próximo número, donde crearemos estos elementos. En la función "Mapa_Colisiones()" del módulo "Fjuego.bb" se encuentra el mapa completo de colisiones para "Zone of Fighters".

¿CÓMO AISLAR LAS COLISIONES?

Una vez determinado el mapa de colisiones de todo el juego debemos establecer las diferentes acciones derivadas del choque entre las entidades. Como sabemos, el mapa de colisiones nos sirve para activar todas las colisiones y obtener una respuesta determinada al choque, pero sólo eso. No sabemos, por ejemplo, con qué entidad en particular hemos colisionado ni podemos determinar una acción específica. Para ello, disponemos de una serie de instrucciones que utilizadas adecuadamente nos ayudarán en esta misión.

Para implementar un desplazamiento suave sobre el terreno es conveniente aplicar el método 3.

Se puede islar una entidad de un grupo del mismo tipo con "CountCollision" y "CollisionEntity".

Código 1. Detectamos con qué "volador" chocamos

```
If EntityCollided (bionave,ENTIDAD_ENEMIGO)
    If escudo_bionave=1 Return   ; Si el escudo está activado queda inmunizado, así que volvemos
    For n=1 To CountCollisions(bionave)
    ent=CollisionEntity(bionave,n) ; Guardamos con qué volador chocamos
    Next
    For volador.tipo_voladores= Each tipo_voladores
        If volador\entidad_volador=ent vida=vida-20 :play(schoque,c_schoque,bionave): Return
    Next
EndIf
```

En la función "actualizar_jugador_principal" del módulo "jugador_principal.bb" es donde resolvemos todas estas cuestiones. Antes de proseguir hay que recordar que los "voladores" son esos cubos que pululan por el escenario y que siempre persiguen a la bionave. En el mapa de colisiones los definimos como del tipo "enemigo", así que vamos a detectar con qué "volador" chocamos (Ver Fig.4) (Ver Código 1).

Una vez que hemos chocado con cualquier "volador" ("tipo ENEMIGO") utilizamos un bucle para recorrer cada una de las colisiones (almacenadas en la lista de colisiones) que ha sufrido la bionave durante la última actualización del mundo 3D con "CountCollisions". La otra entidad involucrada la guardamos en el manipulador "ent", el cual nos servirá posteriormente para averiguar el "volador" en concreto. Así que recorremos la estructura "tipo_voladores" y comparamos cada entidad de la estructura con la almacenada en "ent". Si coinciden, restamos vida a la bionave y volvemos. Hemos aprendido que lo importante es averiguar con qué entidad en concreto hemos colisionado, las consecuencias estarán determinadas por cada situación.

Para detectar los disparos enemigos no hace falta saber con cuál de ellos se ha colisionado ya que para todos se produce la misma respuesta.

Código 2. Aislamos cada tipo de armamento

```
If EntityCollided (disparo\sprite,ENTIDAD_TERRENO )
 Select tipo_armamento
  Case 3          ; Bombas de minifusión
   If Terreno_dinamico=True
    ex#=EntityX(disparo\sprite)
    ey#=EntityY(disparo\sprite)
    ez#=EntityZ(disparo\sprite)
    TFormPoint( ex,ey,ez,0,terreno1 )
    hi#=TerrainHeight( terreno1,TFormedX(),TFormedZ() )
    If hi>0
     hi=hi-.1:If hi<0 Then hi=0
     ModifyTerrain terreno1,TFormedX(),TFormedZ(),hi,True
     ModifyTerrain terreno1,TFormedX()+Rnd(-1,1),TFormedZ()+
     Rnd(-1,1),hi,True
     ModifyTerrain terreno1,TFormedX()+Rnd(-1,1),TFormedZ()+
     Rnd(-1,1),hi,True
    EndIf
   EndIf
   Crear_explosion(disparo)
   FreeEntity disparo\sprite
   Delete disparo
   Return
  Default
 End Select
Endif
```

IMPLEMENTANDO LAS COLISIONES DE LOS DISPAROS

Quizás el proceso más interesante y complejo sea dar a conocer la presencia de los disparos en el juego al resto de entidades. Debemos implementar los procesos de colisiones en la función de actualización de cada disparo "actualizar_disparo" del módulo "jugador_principal.bb", ya que es la mejor forma de controlar las diferentes acciones por cada disparo individualmente.

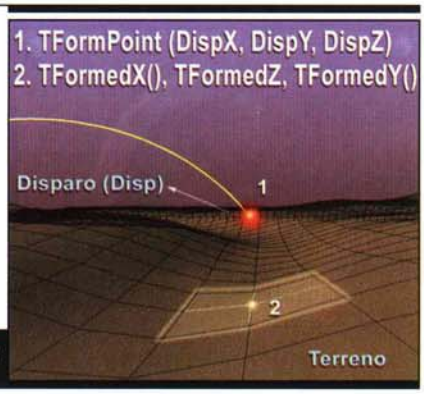
COLISIONANDO Y DEFORMANDO EL TERRENO

Según nuestro diseño inicial, se contempla la posibilidad de deformar el terreno para crear cráteres causados por el impacto de bombas de minifusión (Ver Fig.5).

De este modo, después de detectar la colisión debemos aislar cada tipo de armamento con una estructura "Select.. Case" (Ver Código 2).

5

Con el comando "ModifyTerrain" podemos obtener resultados como el que se muestra en la imagen.



6

Para poder modificar un terreno con "ModifyTerrain" es necesario averiguar las coordenadas exactas donde cayó el disparo por medio de "TFormPoint".

Para modificar el terreno utilizaremos la función "ModifyTerrain", la cual modifica la altura en un punto determinado de éste. Pero, ¿cómo podemos averiguar qué punto modificar? Esta información nos la proporcionarán las coordenadas del propio disparo. No obstante, para que todo surta efecto, es necesario trasladar estas coordenadas a las del terreno, las cuales son almacenadas por la función"TFormPoint".

A continuación, es preciso recuperar estos valores para que puedan ser utilizados por la función "ModifyTerrain". Para tal fin disponemos de las funciones "TFormedX", "TFormedY" y "TFormedZ" (Ver Fig. 6).

Después de formar el cráter, creamos una explosión, borramos la entidad disparo de la memoria y regresamos.

COLISIONANDO CON EDIFICIOS Y ÁRBOLES

En las colisiones con los edificios (almacenes, puentes, generadores y rocas), además de crear la explosión, vamos a dejar una mancha. A este tipo de entidad la denominaremos "agujero" y es un sprite que orientaremos en la dirección del polígono que recibe el impacto.

Esta orientación la realizamos con la instrucción "AlignToVector", la cual alinea los ejes del sprite en la dirección del vector establecido por las normales del polígono impactado. Para poder orientar el sprite en cualquier dirección es necesario establecerle previamente el modo 4 de visualización con "SpriteViewMode" (Ver Código 3).

Código 3. Establecemos el modo 4 de visualización

```
If EntityCollided(disparo\sprite,ENTIDAD_EDIFICIO)
 If Decorado_interactivo=True
  For n=1 To CountCollisions(disparo\sprite )
   colision_x#=CollisionX(disparo\sprite,n )
   colision_y#=CollisionY(disparo\sprite,n )
   colision_z#=CollisionZ(disparo\sprite,n )
   normal_x#=CollisionNX(disparo\sprite,n )
   normal_y#=CollisionNY(disparo\sprite,n )
   normal_z#=CollisionNZ(disparo\sprite,n )
   ;Dejamos una mancha donde impacta el disparo para simular un agujero
   agujero.aguje=New aguje
   agujero\alfa=1
   agujero\sprite=CopyEntity(sprite_agujero)
   ScaleSprite agujero\sprite,Rnd(6,15),Rnd(6,15)
   PositionEntity agujero\sprite,colision_x,colision_y,colision_z
   AlignToVector agujero\sprite,-normal_x,-normal_y,-normal_z,3
   MoveEntity agujero\sprite,0,0,-.1
   Exit
  Next
 EndIf
 Crear_explosion(disparo)
 FreeEntity disparo\sprite
 Delete disparo
 Return
EndIf
```

Otro elemento del decorado que debemos detectar son los árboles. A diferencia de los edificios, no permanecerán estáticos cuando reciban el impacto de un disparo, sino que se moverán aleatoriamente unos pocos píxels en su eje X e Y:

```
If EntityCollided
(disparo\sprite,ENTIDAD_ARBOL)
 For n=1 To CountCollisions
  (disparo\sprite )
   ent=CollisionEntity
(disparo\sprite,n)
   xent#=EntityX(ent)
   yent#=EntityY(ent)
   zent#=EntityZ(ent)
   TurnEntity ent,Rnd(1,4),
   Rnd(1,4),0
 Next
...
```

Como podemos observar todos los procedimientos son prácticamente iguales entre sí.

Con este número hemos completado el estudio de la implementación de nuestra bionave de combate en el juego. El siguiente paso será dar vida al terreno de juego, controlando la manera de colocar todos los elementos del decorado según el sistema de juego seleccionado. Pero antes, sería interesante hacer un inciso para dedicar una entrega al sistema de partículas que nos permitirá crear efectos especiales como fuego o humo.

DEFINICIÓN

▸ VECTOR Y NORMAL

En un espacio 3D, un *vector* es un segmento de recta colocado en cualquier punto y con una dirección determinada. Una *normal* es un vector perpendicular a cualquier polígono.

En el próximo número...

... aplicaremos los efectos especiales de humo, fuego o climatología.

Fabricando los elementos del juego (IV)

Una vez que hemos terminado de crear los seres vivos del juego, pasamos a realizar los elementos estáticos del decorado (almacenes, generador, puente y rocas), así como los árboles pétreos.

MODELANDO LOS ALMACENES

En "Zone of Fighters" disponemos de dos tipos de almacenes, uno cúbico y otro pentagonal. Empezaremos modelando el primero de ellos con Milkshape3D.

Una vez preparadas las vistas pasamos a crear el cuerpo principal de la estructura, la cual basamos en un cubo. Para el techo, añadimos un par de cubos más, pero de menor altura y tamaño el primero y de igual altura pero de mayor tamaño el segundo. Este último cubo servirá para añadir una especie de visera, tal y como se muestra en la figura 1.

Continuamos, modelando y colocando las piezas de las esquinas. Para poder situarlas mejor, es conveniente ocultar el cubo mayor anterior (visera). Así que la seleccionamos en la pestaña *Groups* en el panel de herramientas y pulsamos en *Hide*.

Para las piezas de las esquinas partimos de un cubo escalado en forma de pilar. Para obtener la forma definitiva, seleccionamos los dos vértices exteriores de la parte superior y los unimos a los vértices interiores. Seguidamente, ajustamos la base de la pieza al cuerpo del almacén, desplazando los dos vértices interiores de la base hacia las paredes del cubo principal.

Para finalizar la pieza, unimos los vértices exteriores de la misma base entre sí ("Ctrl" + "N") (Ver Fig. 2).

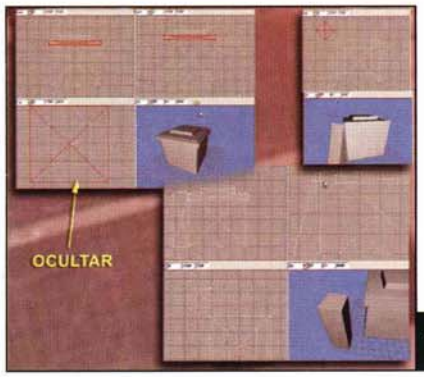
Para crear las otras piezas de las esquinas utilizamos funciones de duplicación y *Mirror*. Para poder continuar, volvemos a mostrar la pieza que ocultamos anteriormente. En ella, seleccionamos los vértices inferiores y los desplazamos para unirlos con la estructura principal del almacén. Terminamos, colocando la chapa en la pared donde irá posteriormente el logotipo del juego mediante un cubo escalado.

Para modelar el segundo almacén partiremos de un cilindro con 1 "Stack" y 7 "Slices" y con la opción *Close Cylinder* activada que servirá de cuerpo principal de la estructura. Para el techo, utilizamos duplicaciones escaladas de este cilindro tal y como se muestra en la figura 3.

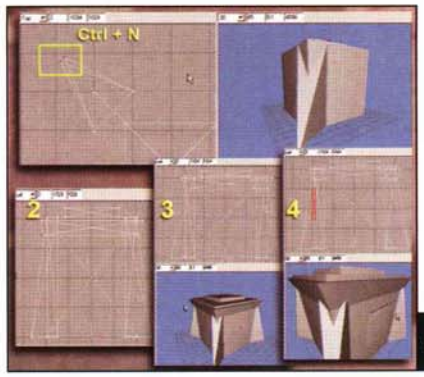
Para terminar el techo, seleccionamos los vértices de la cara superior de la última duplicación y los desplazamos hacia el interior con la herramienta de escalado. Finalizamos, colocando la pieza para situar el logotipo.

MODELANDO UN GENERADOR

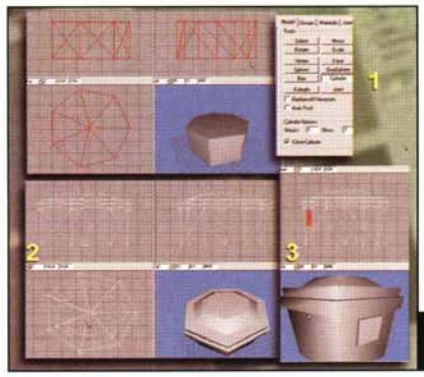
Los generadores son elementos más complejos de modelar. Constan de un cuerpo principal cilíndrico al que se le añade por ambos lados una gran tubería de tres piezas que llega hasta el suelo. Empezando con el cuerpo principal, situamos un cilindro de 1 "Stack" y 8 "Slices" y lo rotamos sobre su eje X 90 grados para volcarlo. Para



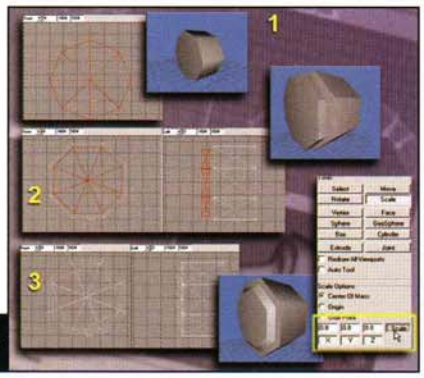
Para poder ajustar correctamente los pilares a cada esquina de la estructura principal es conveniente ocultar la pieza seleccionada de la figura.



Los vértices exteriores de la base del pilar se pueden unir para obtener la forma deseada.

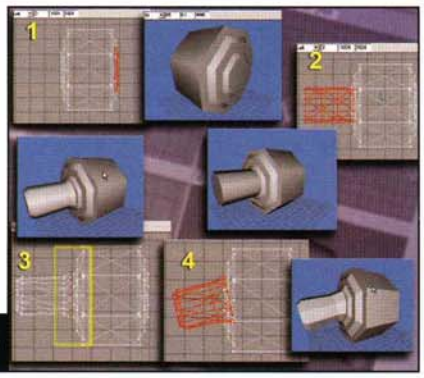


Duplicando y escalando el pentágono principal podemos obtener el resto de elementos del almacén.



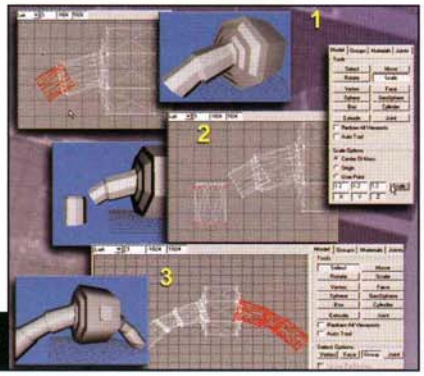
4

Duplicando el hexágono principal y escalando la copia obtenemos el anclaje mayor de las tuberías.



5

Las tres partes del cilindro que forman la primera sección de la tubería nos servirán para establecer la forma de los chaflanes de los extremos. Procedimiento 3 y 4 de la figura.



6

Debemos ajustar y rotar las demás secciones de la tubería para que alcance el suelo.

crear los anclajes de las tuberías utilizaremos el mismo cilindro, duplicándolo y aplicándole un escalado de 0.9 puntos en todos los ejes. De esta forma, nos ahorramos tener que rotar un cilindro nuevo. Seguidamente, seleccionamos los vértices exteriores de esta pieza y los escalamos hacia dentro en proporción 0.8. Realizamos las mismas operaciones para la siguiente pieza más pequeña del anclaje (Fig. 4).

Una vez terminado, seleccionamos las dos piezas y la duplicamos para crear el otro anclaje. Posteriormente, sólo tenemos que aplicar un *Mirror front <-> back* para situarlo al otro lado (Fig. 5 / 1).

El siguiente paso será crear una de las tuberías unidas al anclaje. Esta pieza consta de un cilindro situado en medio de otros dos mayores. Inicialmente, creamos el primer cuerpo de la tubería, el cual está unido a la estructura principal a través de uno de los anclajes. Para ello, partimos de un cilindro cerrado de 3 "Stacks" y 9 "Slices". Los "Stacks" o partes nos servirán para crear los ajustes de los extremos del trozo de tubería (Fig. 5 / 2).

Seleccionamos los vértices de cada parte y los desplazamos uno hacia el lado izquierdo y otro hacia el derecho. Posteriormente, seleccionamos los vértices de cada extremo y los escalamos manualmente hacia fuera (la parte interior) y hacia dentro (la exterior) o bien aplicando valores de 0.8 en cada eje. Obtendremos el resultado de la figura 5 / 3. Finalizaremos, girando un poco esta pieza para ir orientando la tubería hacia el suelo (Fig. 5 / 4). La siguiente parte está situada en el centro de la tubería y es de menor tamaño. Se realiza de la misma manera pero partiendo de un cilindro con sólo un "Stack", ya que no lleva uniones y queda incrustada en las otras dos piezas mayores. Seguidamente, volvemos a aplicar un pequeño giro para seguir la orientación hacia el suelo (Fig. 6 / 1).

Concluimos con la tercera parte de la tubería, que terminará introducida en la tierra. Por este motivo, sólo tendrá dos "Stacks", ya que sólo posee una unión en la parte interior (Fig. 6 / 2). Hasta el momento tenemos sólo una de las tuberías completa. Para realizar la del otro lado, como siempre, únicamente tenemos que seleccionarla por completo, duplicar y realizar un *Mirror front <-> back*. Y como es habitual, colocaremos un cubo escalado en el cuerpo principal para situar el logotipo del torneo (Fig. 6 / 3).

MODELANDO UN PUENTE

Esta pieza es un poco más compleja que la anterior y consta de dos partes diferenciadas: la base y techo del puente y los soportes del techo.

BASE DEL PUENTE

Esta parte de la estructura consta de tres piezas cúbicas. Situamos la primera que servirá de estructura principal donde estarán unidas todas las demás. Las siguientes dos piezas están colocadas justo bajo la principal y tienen la misma forma pero con menor tamaño. Para la segunda, haremos una duplicación y aplicamos un escalado con relación 0.9 en el eje X, 1.5 en el Y y 0.95 en el Z. La tercera pieza es una copia de la segunda pero con los vértices exteriores escalados hacia dentro, aplicando el valor 0.95 en cada eje (Fig. 7 / 1).

TECHO Y SOPORTES

Los soportes requieren un poco más de trabajo. Consiste en la consecución de pequeñas piezas cúbicas ligeramente inclinadas para formar un soporte curvo (Fig. 7 / 2 y Fig. 8 / 1).

Posteriormente, debemos unir cada uno de estos pequeños cubos entre sí. Para tal fin, uniremos los vértices superiores de la pieza de abajo con los vértices inferiores de la pieza de arriba, y así sucesivamente hasta unirlas todas (Fig. 8 / 2). El siguiente paso será realizar el acabado del soporte, estrechando los vértices superiores del extremo hasta conseguir el aspecto de la figura 9 / 1.

Los demás soportes se realizan por el procedimiento habitual de duplicado y aplicación de "mirrors" (Fig. 9 / 2).

Para terminar el puente sólo nos queda el techo, el cual es exactamente igual que la pieza principal del puente; es decir, un cubo que escalaremos hasta encajar con los cuatro soportes (Fig. 9 / 3).

MODELANDO LAS ROCAS

Estas estructuras adquieren tres formas diferentes, así evitamos feas repeticiones en elementos agrupados. Todas ellas se modelan de la misma manera, mediante la modificación de los vértices de un cilindro con más o menos partes y divisiones. Para la primera de las rocas partiremos de un cilindro cerrado con 2 "Stacks" y 8 "Slices" (Fig. 10 / 1).

A mayor número de partes y divisiones, mayor número de polígonos y complejidad de formas. Situándonos en la vista superior ("Top") seleccionamos todos los vértices de una división y los desplazamos hacia fuera. Hacemos lo mismo con la división adyacente (Fig. 10 / 2). De esta forma alargaremos la estructura de forma irregular. Para resumir, obtendremos muy diversas formas manipulando cada vértice o grupo de ellos en cada una de las divisiones. Podemos utilizar procedimientos de rotación, escalado o desplazamiento pero siempre siguiendo un orden y evitando la rotación excesiva de polígonos, ya que esto provocaría que sus caras no sean visibles (Fig. 10 / 3).

Para construir el resto de las rocas utilizaremos los mismos procedimientos. En estos casos es la imaginación en la manipulación de vértices la que origina las diferentes formas.

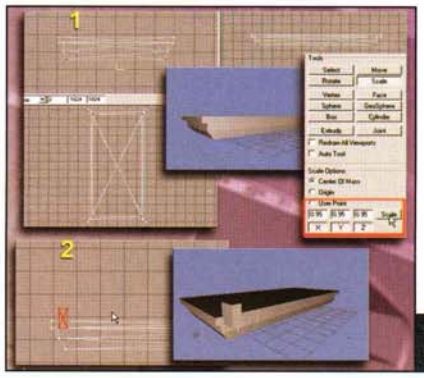
MODELANDO ÁRBOLES PETRIFICADOS

La creación de troncos para árboles es partícipe de la misma idea que utilizamos en el modelado de rocas. En este caso, podemos partir de un cilindro con 4 "Stacks" y 10 "Slices" (Fig. 11 / 1).

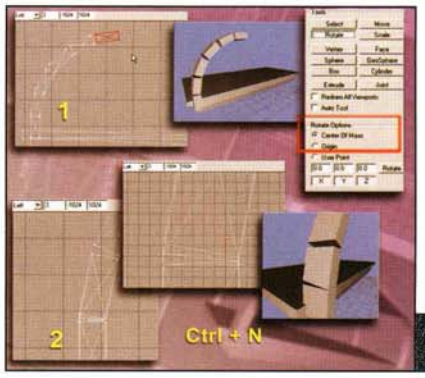
La idea radica en que el número de partes influye en la obtención de una forma de tronco más o menos compleja. Modificando los vértices de cada una de estas partes, por ejemplo, aplicando la herramienta de escalado, obtendremos formas variadas como troncos retorcidos. Lo primero que debemos modificar para crear un árbol son los vértices del primer "Stack" (parte), los cuales nos ayudarán a formar la base del árbol. El resto de vértices darán la forma del tronco (Fig. 11 / 2).

Para crear las ramas podemos realizar las mismas operaciones pero en cilindros más pequeños, los cuales uniremos al tronco posteriormente (Fig. 11 / 3).

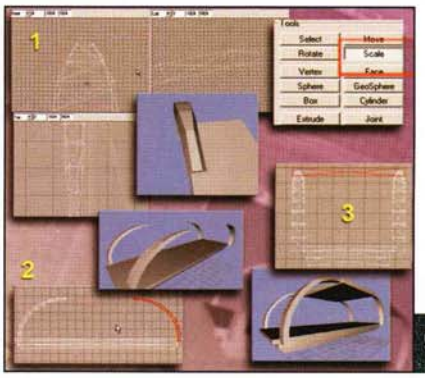
Después del modelado, el siguiente paso en la creación de un objeto 3D para un juego es el texturizado del mismo. Para las estructuras que hemos modelado en esta entrega, utilizaremos dos tipos de procedimientos de texturizado: pintando directamente en el modelo con Deep Paint 3D (generador) y por medio de plantillas ("templates") en un programa de dibujo como Paint Shop Pro (almacenes, puente, rocas y árboles). Dejaremos este último



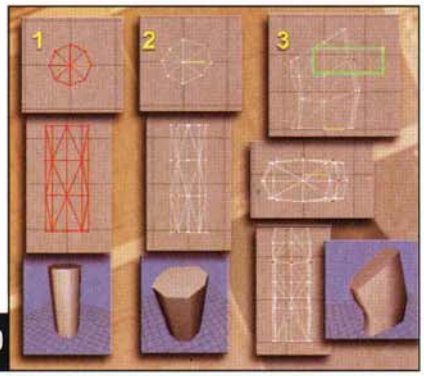
La base del puente consta de tres piezas iguales unidas, pero con diferente escalado.



Una vez colocadas consecutivamente las secciones del soporte, se unen a través de los vértices. De esta manera es fácil obtener formas curvas con pocos polígonos.

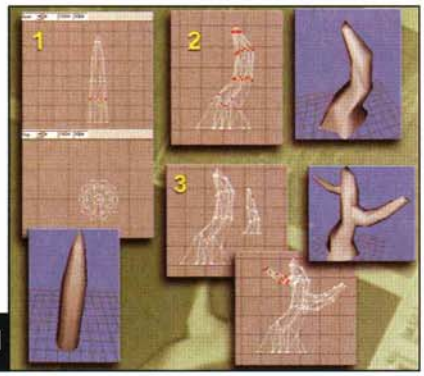


Mediante el escalado de vértices pares podemos estrechar estructuras.



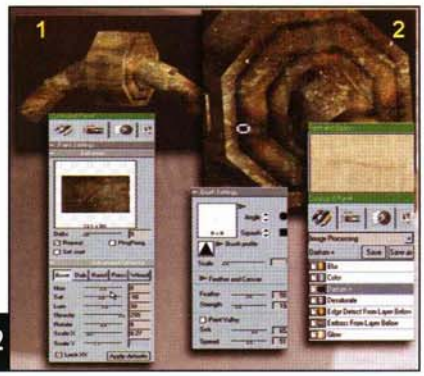
10

A partir de un cilindro con más de tres partes se pueden obtener rocas de diferentes formas desplazando sus vértices.



11

La forma de modificar la posición de los vértices en cada sección de un cilindro determinará las diferentes estructuras.



12

Obtendremos más sensación de volumen en nuestro modelo si perfilamos con el pincel las uniones de sus partes con tonalidades oscuras.

procedimiento para el siguiente número del curso y nos centraremos ahora en el texturizado del generador.

CREANDO EL MAPEADO UV DEL GENERADOR CON LITHUNWRAP

Como es habitual, antes del proceso de texturizado debemos crear un mapeado UV del modelo. Para el generador utilizaremos LithUnwrap. En primer lugar, cargamos el modelo salvado por Milkshape3D (.MS3D o .OBJ) (si no está a mano, se puede encontrar en "Extras" del CD llamado "generador.ms3d" o "generador.obj") Seguidamente, aplicamos un optimizado en *Tools/ Optimize models* para reducir polígonos. Seleccionamos todo con *Select all* y aplicamos un *UV Mapping Planar* con la opción *Planar X*. Terminamos el proceso, salvando de nuevo el modelo en formato .OBJ como "generador.obj".

TEXTURIZANDO EL GENERADOR CON DEEP PAINT 3D

Comenzamos cargando el modelo anterior. Al hacerlo aparecerá la ventana *Material Import*, indicando la presencia de un mapeado UV en el modelo y pidiendo la creación de un material nuevo para poder pintar sobre el modelo. Lo crearemos y seleccionaremos para él un tamaño de 256 x 256. Activamos el canal *C* con *nothing color* y ya estamos preparados.
Continuamos, asignando como vista de trabajo *Top* en *View*. Pulsamos dos veces seguidas sobre el icono de la herramienta de la lupa para ajustar el modelo a la ventana (para que surta efecto, debemos agrandar un poco manualmente el tamaño de la ventana de trabajo). Para rellenar el modelo seleccionaremos la textura *Metal Old 1 +* en la sección de texturas *Texture Paint*, a la que le modificaremos la saturación de color con un valor de -16 en *Sat*, la luminosidad a 30 en *Lum* y un escalado del eje X a 0.27 en *Scale X* (Fig. 12 / 1).

Una vez rellenado el modelo, vamos a utilizar el pincel para perfilar las uniones de las piezas del generador para distinguirlas. Aplicaremos un efecto de oscurecimiento de la imagen con pincel de tamaño 5. Este proceso lo seleccionamos de la sección *Imagen Processing* y escogemos *Darken +* (Fig. 12 / 2).

Una vez terminado el proceso de pintado, vamos a salvar la textura final. Así que nos dirigimos a la sección de materiales y pulsamos el botón derecho del ratón sobre la capa de color *C* y elegimos la opción *Export Channel*.

NOTA

**Un buen ejercicio para practicar el modelado con Milkshape3D sería realizar nuevos edificios, rocas o árboles o modificar los existentes.
Posteriormente, podríamos pintarlos con Deep Paint 3D y utilizar nuestro propio visualizador de objetos para ver los resultados, el cual desarrollamos en la sección de desarrollo del número 7 del curso. Incluso se podrían sustituir los ficheros del juego por los nuevos (siempre que se mantengan los mismos nombres) y de esta forma ver nuestros modelos en el editor de zonas de combate o en el propio juego.**

En el próximo número...

... realizaremos las texturas de los demás elementos del decorado por medio de plantillas con Paint Shop Pro.

Nuestro primer tema musical con Anvil Studio (II)

Continuando con la serie dedicada a Anvil Studio, comenzamos en este número el estudio de cómo realizar nuestro tema musical.

Como sabemos, en Anvil Studio podemos crear música mezclando pistas MIDI, de audio y de ritmos. Las pistas MIDI se pueden grabar utilizando un instrumento externo como un sintetizador, pero si no disponemos de ninguno, es posible hacerlo manualmente desde el *Compose* colocando nota a nota sobre el pentagrama. Para grabar pistas de audio, podemos utilizar un micrófono o a través de la entrada de línea de la tarjeta de sonido. Y para crear una pista de ritmo, sólo tenemos que elegir un sonido MIDI de la lista o insertar cualquier muestra.

CREANDO Y GRABANDO EN UNA PISTA MIDI

Vamos a crear una nueva canción, empezando por la grabación de una pista MIDI utilizando un instrumento externo conectado al ordenador a través de las conexiones MIDI de la tarjeta.

Por defecto, el programa se ejecuta con una pista de instrumento (MIDI) creada. Así que sólo tenemos que seleccionarla para poder grabar en ella. Pero antes es necesario tener las conexiones correctamente. Para más detalles en la figura 1 se muestra una conexión típica de hasta dos teclados al ordenador.

A continuación, debemos asegurarnos de que el programa recibirá los eventos MIDI desde el teclado externo. Para ello, nos vamos al menú *View* y entramos en *Synthesizers*, en el cual se muestran los puertos de entrada y salida de audio y MIDI. En la casilla *MIDI In Port* debe aparecer *Maestro MPU-401* o *MIDI In*. Si queremos que suene el sintetizador externo debemos colocar en *MIDI Out Port* lo mismo que en el puerto de entrada. Si queremos que suene la tarjeta del ordenador elegimos los mapeadores MIDI o el sintetizador de la tarjeta de sonido. (Fig. 2 / 1).

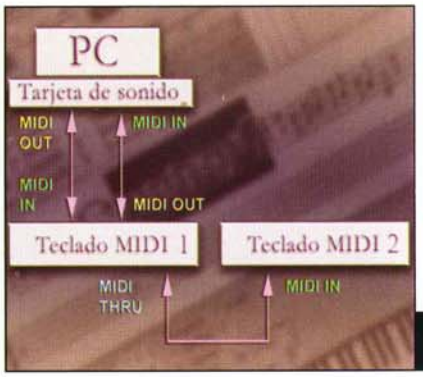
Es posible averiguar si todas las conexiones son correctas pulsando en el botón "Test MIDI Connections" (Fig 2 / 2).

Antes de grabar podemos también averiguar si todo marcha bien a través de la representación gráfica de luces que aparecen en el *Mixer* al lado del transportador. Estas "lucecitas" representan la actividad MIDI. Cada una de ellas interpreta un tipo de actividad cuando se torna de color rojo. En la figura 3 se muestra una tabla con el significado de cada una de estas actividades.

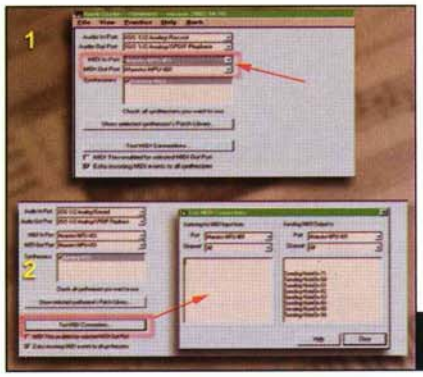
Bien, ya está todo preparado para grabar. Simplemente, pulsamos el botón de grabación "REC" y el programa hará una cuenta atrás antes de empezar a grabar. Sin embargo, antes de cualquier grabación es necesario preparar el tiempo y el compás, así como el metrónomo para acomodar la grabación a nuestras necesidades.

TIEMPO, COMPÁS Y METRÓNOMO

Para modificar los parámetros iniciales de nuestra canción, como la velocidad o el metrónomo, debemos entrar en la ventana de opciones



Esquema de una conexión MIDI de hasta dos teclados externos.

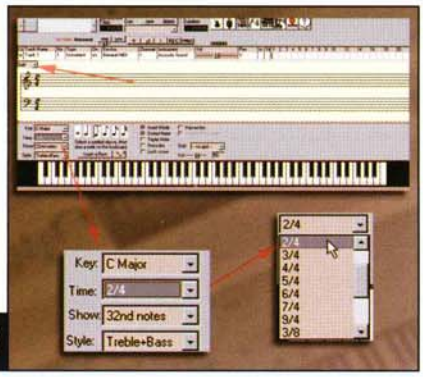


En Synthesizers del menú View podemos ajustar el puerto MIDI y realizar un chequeo de las conexiones.

Significado de la representación gráfica del estado de las conexiones MIDI.

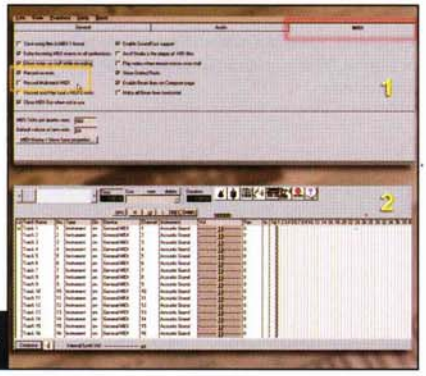
4

Descripción de la ventana de opciones del metrónomo y tiempo de la canción.



5

Procedimiento para cambiar el compás de una pista.



6

En las opciones MIDI del panel View / Options podemos activar la grabación MIDI simultánea en varias pistas.

Metronome / Tempo settings en el menú *View*. En primer lugar, encontramos la casilla *Tempo*, donde elegiremos el tiempo en pulsos por minutos. Seguidamente, las siguientes opciones se refieren al metrónomo. Resaltamos la posibilidad de cambiar su volumen o el sonido según un canal MIDI o nota determinada. En la figura 4 se muestra una descripción de esta ventana.

Para modificar el compás debemos hacerlo en la sección *Compose* en la forma de representación *Staff*. Una vez situados en esta sección podemos elegir el compás en la casilla *Time* (Fig. 5).

OPCIONES DE GRABACIÓN MIDI

Hay varias opciones muy interesantes que podemos activar para cambiar los aspectos de la grabación (Fig. 6 / 1).

Por ejemplo, se puede grabar en varias pistas MIDI a la vez. Para ello, nos situamos en la pestaña *MIDI* en la sección de opciones *View / Options*. Al volver al *Mixer* después de activarla observamos cómo automáticamente se han creado 16 pistas, una para cada canal MIDI, y todas con el mismo instrumento (Fig. 6 / 2). Al tocar, todos los eventos MIDI se grabarán por igual en todas las pistas.

Si estamos grabando desde *Compose* en el pentagrama *Staff*, es posible visualizar las notas al mismo tiempo que grabamos. Esta opción está situada también en las opciones *MIDI* y se denomina *Show notes on staff while recording*.

Una opción muy interesante se nos presenta también en las opciones MIDI denominada *Record No Rests*. Activándola, permitimos al programa que grabe exactamente como estamos tocando sin ningún tipo de cuantización en tiempo real. Si grabamos con esta opción activada es fácil que no ajustemos las notas a los compases perfectamente. En tal caso, siempre podemos aplicar una cuantización nosotros mismos. Para ello, seleccionamos la opción *Quantitize Entire Track* en el menú *Track*. Aparecerá una ventana flotante con la medida de cuantización.

OPCIONES ESPECIALES

Anvil Studio posee posibilidades muy interesantes a la hora de reproducir nuestras canciones. Por ejemplo, permite que el contenido de una pista suene con dos instrumentos a la vez. Evidentemente, para que esto sea posible, es necesario duplicar la pista y cambiar el canal MIDI. Una vez realizado, cambiamos el sonido de la pista duplicada. Para hacer más rápido este procedimiento, disponemos de una opción que divide la pista, creando una nueva idéntica. Generalmente se usa cuando tenemos una canción en una sola pista. Esta opción la podemos encontrar en el menú *Track* y se denomina *Split track into single channel track*. Como disponemos de hasta 16 canales MIDI, podemos tocar una misma pista con 16 instrumentos diferentes a la vez.

Para terminar, Anvil Studio admite posibilidades de transportación de notas para una sola pista o para la canción completa actuando en todas las pistas a la vez.
Encontramos esta opción en el menú *Track / Transpose*. En ambas posibilidades el programa nos pedirá la cantidad de medios tonos que queremos transportar. De esta forma, con un valor de 12 subimos una octava completa y con uno de -12, la bajamos.

Hasta aquí nuestros primeros pasos para crear una canción.

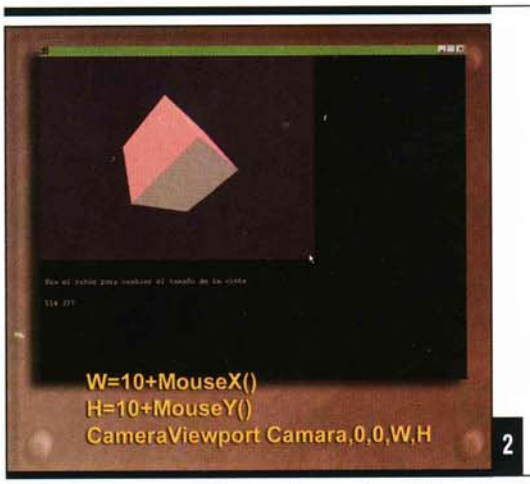
En el próximo número...

... aprenderemos cómo editar en *Compose* entre otras cosas interesantes.

Manejo de cámaras con Blitz3D

Después de conocer, en el número anterior, la plasticidad que ofrece la animación de modelos en un mundo 3D, vamos a descubrir, con esta entrega, la definición de la cámara y sus posibilidades en Blitz3D.

Es importante entender el concepto de cámara en un mundo 3D. En 2D no es necesario este elemento, porque la imagen que vemos se extiende sólo en dos dimensiones; es decir, una superficie plana distribuida en los ejes X e Y. Por lo tanto, la cámara se mantiene fija en una misma posición sin posibilidad de cambio. Por el contrario, en 3D disponemos de un eje más, Z, con lo que es posible girar alrededor de la vista o cambiar la posición de la cámara. En 2D, la cámara se crea y coloca por defecto siempre en el mismo lugar; pero en 3D, necesitamos hacerlo nosotros mismos. En Blitz3D, la cámara se considera una entidad más y, además de mostrar el dibujado de la escena, nos permite controlar otros factores como el rango de visión, el "zoom" o efectos de niebla.



Con la función "CameraViewport" se puede modificar el tamaño horizontal y/o vertical de la vista de cámara.

NOTA

Es importante saber que la cámara sólo funciona en el búfer oculto (backbuffer).

DEFINICIÓN

▸ VIEWPORT

Una vista (viewport) es un área de la pantalla que corresponde a lo que la cámara ve .

CREANDO Y MANIPULANDO LA CÁMARA

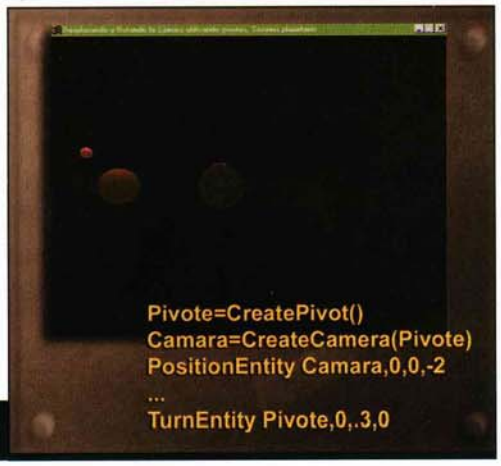
Una vez establecido el modo gráfico, lo siguiente que debemos hacer para preparar nuestra escena 3D es crear al menos una cámara para visualizarla. Para realizar esta operación disponemos del comando "CreateCamera":

```
Camara = CreateCamera
( [entidad madre])
```

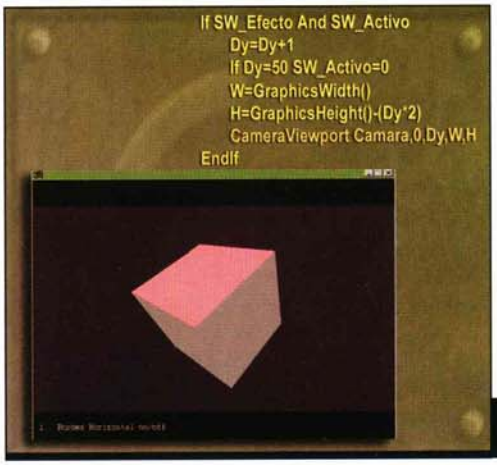
El parámetro opcional nos permite encadenar la cámara a una entidad cualquiera, por lo que si ésta se mueve, la cámara lo hará con ella.

Es evidente que el hecho de que Blitz3D trate a la cámara como una entidad más nos posibilita la aplicación de instrucciones de movimiento y rotación propia de entidades como "MoveEntity", "RotateEntity" o "TurnEntity". Así que podemos mover la cámara por un escenario, seguir a nuestro personaje en un juego o desplazar la cámara alrededor de un objeto por medio de la utilización de pivotes (Ver "ejemplo1.bb") (Fig. 1).

Otra opción muy interesante es que podemos crear cuantas cámaras queramos al mismo tiempo, con lo que obtendremos



La cámara es tratada como una entidad más, así que es posible desplazarla y rotarla con las funciones estándar como "MoveEntity" o "TurnEntity".



En la figura se muestra un efecto 16:9 estrechando la vista de la cámara.

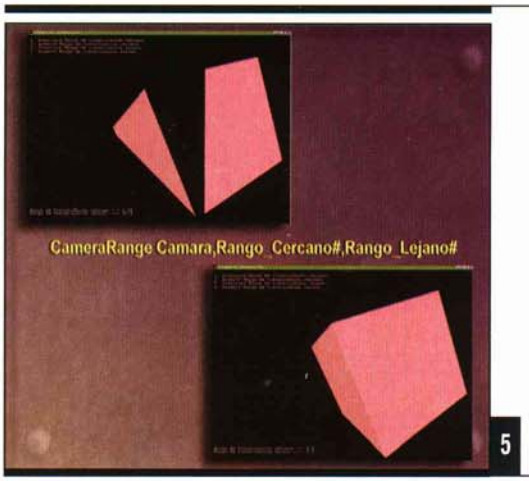
4

En Blitz3D podemos disponer de múltiples cámaras, cada una con una vista y efectos diferentes.

```
GraphicsWidth(),
GraphicsHeight()
```

De esta forma, si queremos tener nuestro mundo 3D visible exclusivamente en la mitad superior de la pantalla sólo tendremos que definir el "viewport" como sigue:

```
CameraViewport
camara, 0,0,
GraphicsWidth(),
GraphicsHeight()/2
```



5

Mediante la función "CameraRange" podemos definir el rango de visualización de la cámara.

diferentes vistas en la pantalla de una misma escena.

CONTROL DE MÚLTIPLES VISTAS

Por defecto, toda la pantalla es considerada como la vista o "viewport" de la cámara; es decir, desde 0,0 hasta el tamaño horizontal y vertical que marque la resolución activa.

Sin embargo, el "viewport" puede ser modificado a voluntad para que ocupe un área determinada. Para ello, disponemos del comando "CameraViewport":

```
CameraViewport camara, X, Y,
Ancho, Alto
```

Así, la pantalla completa estará definida de la siguiente manera:

```
CameraViewport camara, 0,0
```

DEFINICIÓN

▶ EL Z-BUFFER

El Z-BUFFER es una lista de números que almacena las propiedades de cada píxel de un polígono y su distancia a la cámara. Se utiliza en la técnica "Z-Buffering" para dibujar únicamente los polígonos que se ven. Otro utilidad del "Z-buffering" es la creación de efectos de niebla.

En el ejemplo 2 ("ejemplo2.bb") se puede observar cómo es posible cambiar el tamaño de la vista de la cámara en tiempo real (Fig. 2 y 3).

Blitz3D admite la posibilidad de crear varias cámaras a la vez; con ello se permite manipular más de una vista diferente. Un ejemplo de la aplicación de este sistema lo encontramos en programas 3D, en los cuales se muestran cuatro vistas diferentes de la escena (derecha, izquierda, superior e inferior). También se utiliza mucho en juegos de velocidad, donde disponemos de una vista para simular el espejo retrovisor del vehículo. Las utilidades son numerosas y Blitz3D no nos pone ninguna limitación al respecto, ya que podemos crear 2,10, 20 o más cámaras diferentes, cada una con su propia vista y efectos. La única barrera son las propias limitaciones del sistema. Evidentemente, la utilización de más de una cámara reduce el rendimiento general del programa. En el ejemplo 3 ("ejemplo3.bb") se muestra cómo se pueden crear y manipular a la vez numerosas cámaras cuyos manipuladores pertenecen a una estructura de datos (Fig. 4).

OPCIONES DE VISUALIZACIÓN

En ocasiones, necesitaremos controlar el rango de visualización de nuestra escena, por ejemplo, para limitar hasta cuánto podemos ver. Esto nos permitirá optimizar el rendimiento del "Z-Buffering". En Blitz3D podemos controlar este aspecto con la instrucción "CameraRange":

```
CameraRange Camara,
distancia_cercana#,
distancia_lejana#
```

El Z-BUFFER es una lista de números que almacena las propiedades de cada píxel de un polígono y su distancia a la cámara. Se utiliza en la técnica "Z-Buffering" para dibujar únicamente los polígonos que se ven. Otro utilidad del "Z-buffering" es la creación de efectos de niebla.

El parámetro"distancia_cercana" indica dónde comenzarán a dibujarse los objetos 3D enfrente de la cámara y "distancia_lejana" hasta dónde se dibujarán. Por defecto, se dispone de los valores 1, 1000; es decir: *CameraRange camara,1,1000.* Todos los objetos colocados antes y después de estos valores serán eliminados de la escena ("Clipping"). Controlando estos valores podemos aumentar o disminuir el rendimiento del programa, ya que determinamos la cantidad de polígonos que se dibujarán (Fig. 5, "ejemplo4.bb").

Sin embargo, el "Clipping" provoca un corte en la imagen que no es del todo atractivo a la vista. Para evitar el más cercano

6

Situar la niebla antes del rango de visualización de la cámara ayudará a disimular el efecto "clipping".

se suele aplicar un valor en "distancia_cercana" lo más pequeño posible, como por ejemplo, 0.1, y para disimular el corte más lejano se aplica una efecto de niebla a una distancia inferior al valor "distancia_lejana". Para crear y utilizar un efecto de niebla determinado son necesarios varios pasos. En primer lugar, debemos definir el rango de visualización de la niebla; es decir, a qué distancia, frente a la cámara, estará situada. Seguidamente, le aplicaremos un color, y para terminar es necesario activar su visualización.

Para cada unos de estos procedimientos es primordial indicar a qué cámara se aplicarán, ya que, en caso de utilizar más de una a la vez, llevarán su propio "set" de efectos:

```
CameraRange Camara,0.1, 1000
CameraFogRange Camara, 0.1, 900
CameraFogColor Camara, 0, 0, 0
CameraFogMode Camara, True
...
```

En el procedimiento anterior, se puede observar que la niebla se sitúa antes que el límite de dibujado para evitar, de este modo, visualizar el "Clipping". La instrucción "CameraFogColor" define un color para la niebla:

```
CameraFogColor Camara,
componente de color
rojo#, verde#, azul#
```

y "CameraFogMode" le indica al Blitz3D que la visualice:

```
CameraFogMode Camara,
True/False
```

(Fig. 6, "ejemplo5.bb").

Otras opciones muy interesantes para controlar el visualizado de la cámara consisten en aplicar un color de fondo de diferentes maneras y modificar el factor del "Zoom" (alejamiento y acercamiento). Si queremos borrar la vista con un color utilizaremos "CameraClsColor":

```
CameraClsColor Camara,
componente de color rojo#,
verde#, azul#
```

También podemos seleccionar el tipo de borrado que Blitz3D aplicará a la vista. Por un lado, podemos borrar el color y por otro los objetos 3D. Por defecto, los dos parámetros están activados:

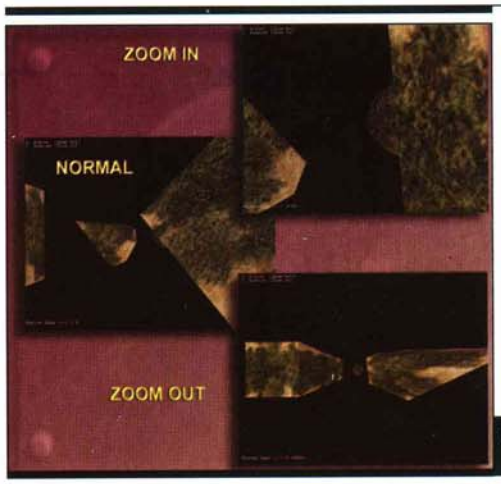
```
CameraClsMode Camara,
Borrado_Color (True/False),
Borrado_Zbuffer (True/False)
```

En el ejemplo 4 se muestra el resultado de cambiar estos parámetros.

Para modificar el "Zoom" disponemos del comando "CameraZoom":

```
CameraZoom Camara, factor_zoom#
```

El parámetro "factor_zoom" es muy sensible y por defecto se sitúa en 1 (Ver "ejemplo5.bb"). Si aplicamos un valor entre 0 y 1 la cámara hará un "zoom out" (alejamiento) y si aplicamos valores superiores a la unidad un "zoom in" (acercamiento) El cambio del zoom de la cámara se conoce también como "FOV" (Fig. 7, "ejemplo6.bb").



7

Podemos modificar el FOV de la cámara con la instrucción "CameraZoom".

FUNCIONES ESPECIALES

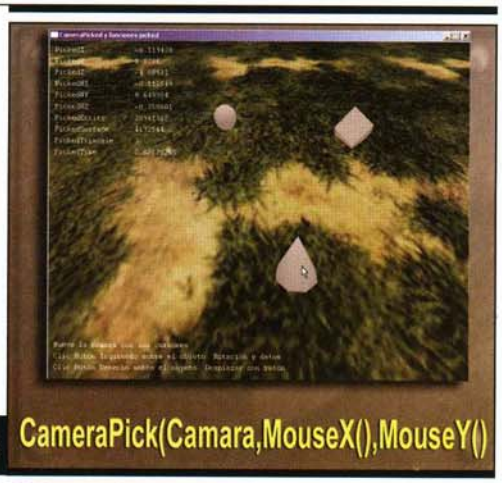
La comunicación entre los sistemas de coordenadas 2D y 3D es fundamental para una correcta implementación de la cámara en numerosas aplicaciones. Blitz3D nos proporciona todas las herramientas necesarias para este cometido. Así que podemos, por ejemplo, situar texto o elementos 2D en una escena 3D o permitir la interacción del ratón con objetos 3D, etc.

Imaginad que tenemos una escena 3D delante de la cámara compuesta de un terreno y tres objetos diferentes sobre él: un cubo, un cono y una esfera. Nuestro propósito es seleccionar cualquiera de estos objetos (incluido el terreno) mediante el puntero del ratón. Para ello, debemos utilizar dos funciones, una para los objetos que queremos seleccionar y otra para encontrarlos. Así que disponemos de la función "CameraPick", la cual nos dice con qué entidad (objeto) ha tocado el puntero del ratón:

```
CameraPick ( Camara,
Coordenada X# del ViewPort,
```

NOTA

No todas las tarjetas gráficas soportan efecto de niebla, y las que sí, ofrecen una calidad diferente.



8

Mediante la función "CameraPick" es posible la interacción entre el puntero del ratón y el entorno 3D.

```
Coordenada Y# del ViewPort)
```

Pero para que esto funcione realmente, antes necesitamos decirle al Blitz3D qué objetos están dispuestos para ser detectados. Por lo tanto, debemos convertirlos en "pickeables"; es decir, que puedan ser tocados con el puntero del ratón. Así que es necesario aplicarles la función "EntityPickMode" con un valor de geometría que no sea 0 (Ver número 8 del coleccionable: "Blitz3D.Manejo de entidades"). Por ejemplo:

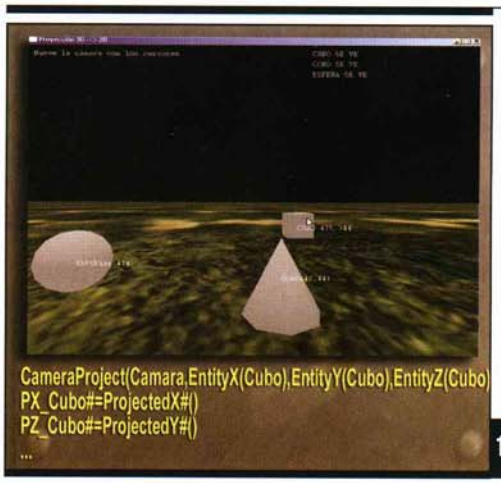
```
EntityPickMode Cubo, 2
```

(Fig. 8, "ejemplo7.bb")

Una vez detectado el objeto 3D, podemos obtener valiosa información sobre su posición y estado. Por medio de las funciones "PickedX#()", "PickedY#()" y "PickedZ#()" obtenemos las coordenadas X, Y y Z exactas donde tocó el puntero. Y con "PickedNX#()", "PickedNY#()" y "PickedNZ#()" obtenemos las componentes de las coordenadas X, Y,y Z de la normal.

Además de las coordenadas, es posible averiguar hasta niveles más inferiores y precisos. Si tocamos con el puntero del ratón el cono de nuestro ejemplo, es posible, incluso, saber qué triángulo o superficie de la estructura del objeto hemos tocado. Esto es posible gracias a las funciones "PickedSurface()" y "PickedTriangle()". En ocasiones, podríamos necesitar saber cuánto tiempo se ha empleado en calcular cualquiera de estos comandos. Es muy útil, por ejemplo, para el testado y optimizado del código cuando estemos trabajando con este tipo de operaciones. Para averiguarlo, disponemos de la función "PickedTime()".

Volviendo a nuestra escena 3D anterior, queremos saber cuáles de los tres objetos (cubo, cono y esfera) quedan fuera de la vista de la cámara y cuáles no. Podemos averiguar esta información con la función "EntityInView". Esta función devolverá un 1 (True) si el objeto es visible y 0 (false) en caso contrario. Hay que tener en cuenta el punto de referencia que utiliza Blitz3D para realizar estas comprobaciones. Así, si nos referimos a un *mesh*, tomará como referencia la caja que lo limita y si es otra entidad, tomará como referencia su centro de posición (Fig. 9).



10

Dependiendo de si es un mesh u otra entidad, la función "EntityInView" tomará distintas referencias del visualizado.

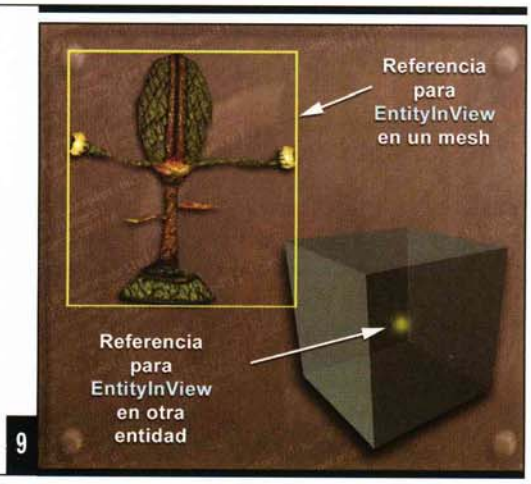
Por último, una interesante opción que encontramos entre las funciones de Blitz3D es la posibilidad de proyectar las coordenadas de un objeto 3D a coordenadas 2D. Esta opción es tremendamente útil, por ejemplo, en aplicaciones donde queremos imprimir un texto o número en la posición de un objeto 3D situado en cualquier lugar de la escena. Así que disponemos para ello de la función "CameraProject":

```
CameraProject Camara,
Coordenada X# (mundo 3D),
Coordenada Y#, Coordenada Z#
```

Esta función pasará los diferentes valores de coordenadas a las funciones "ProjectedX#()", "ProjectedY#()" y "ProjectedZ#()".

En el ejemplo 8 se puede observar el funcionamiento de estas funciones (Fig. 10).

La cámara es uno de los factores primordiales en cualquier escena 3D. Existe, sin embargo, otros cuyo estudio no deja de ser fascinante: la iluminación.



9

Ejemplo del funcionamiento de los métodos de proyección de coordenadas.

En el próximo número...

... estudiaremos el manejo de luces.

Utilización de imágenes para crear fuentes de letras

Como prometimos, vamos a desarrollar una serie de funciones que nos permitirán escribir letras o números en pantalla utilizando caracteres procedentes de imágenes predefinidas.

En primer lugar, debemos crear un alfabeto en una aplicación de dibujo como, por ejemplo, Paint Shop Pro. Nuestro alfabeto estará formado por letras de diseño; es decir, con los colores o formas que queramos. Seguidamente, aislaremos cada carácter y crearemos un fichero de imagen por cada uno de ellos en un formato cualquiera. Proponemos como mejor opción .PNG, ya que es un formato comprimido (como .JPG) y mantiene la calidad del .BMP. Además, posee canal alfa como los .TGA. Es un formato muy completo (Ver Fig. 1).

Cada uno de estos ficheros tendrá el nombre del carácter que corresponde; es decir, para la imagen de la letra "A" el fichero se llamará "A.png", y así sucesivamente.

Luego, en el programa, sustituiremos el carácter alfanumérico por su imagen correspondiente. De esta manera, podemos imprimir frases o números utilizando nuestro alfabeto de diseño sin estar limitados a las fuentes que tengamos instaladas en nuestro sistema.

Para aprovechar este procedimiento, crearemos una serie de funciones que utilizaremos para cada uno de los siguientes casos: imprimir una frase, imprimir una variable numérica o crear una entrada de datos o "Input". Todos ellos servirán perfectamente para implementar la impresión de nuestros caracteres especiales en cualquier situación de un juego: menú, diálogos, contadores (puntuación, tiempo, etc.) o entrada de datos (nombre del jugador, récord, etc.)

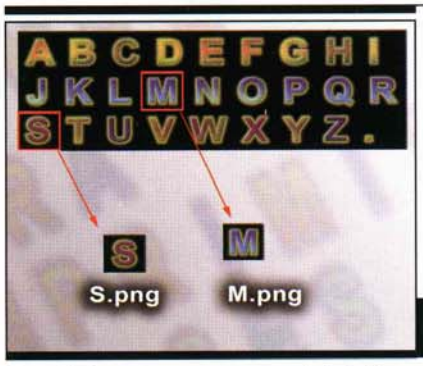
DEFINICIÓN DE DATOS Y FUNCIONES

Lo primero que debemos hacer es crear una matriz con los datos de cada letra del alfabeto de la "A" a la "Z" y de los números del "1" al "0" y otra equivalente para los gráficos correspondientes a cada letra. Pero antes, vamos a definir una constante para el número de caracteres de nuestro alfabeto:

```
Const Num_letras%=35 ; (36 ‡ 0 - 35)
```

Seguidamente creamos ambas matrices:

```
Dim Letra$    (Num_letras)
Dim Grafico (Num_letras)
```



Procedimientos para preparar un alfabeto de diseño.

Continuamos, rellenando la matriz de comparación con todas las letras de nuestro alfabeto a partir de un banco de datos. Aprovechamos también el bucle de lectura de datos para almacenar en la segunda matriz de gráficos las imágenes de cada carácter (Ver Código 1).

De esta forma, tenemos, por ejemplo, en "Letra$(0)" la letra "A" y en "Grafico(0)" la imagen de esta letra "A.png" y así sucesivamente. Después de cargar cada imagen la escalamos a un tamaño definido por las variables globales "SizeV#" y "SizeV#" con la instrucción "ScaleImage". De esta forma podemos controlar también el tamaño de las letras. Recordad que las variables globales anteriores tienen que ser definidas en el programa

TRUCO

La mejor manera de trabajar con las imágenes de nuestro juego es utilizar el formato .BMP. Una vez obtenidos los resultados finales, se pasarán a .PNG. Ahorraremos mucho espacio en disco y, llegado el momento, se tardará menos tiempo en su carga.

Código 1. Almacenamos los caracteres

```
Restore Datos_letras  ;Apuntamos a la base de datos
For n=0 To Num_letras
  Read Letra$(n)
  Grafico(n)=LoadImage(Letra$(n)+".png")
Next
; ------------------------------------------------------------
.Datos_letras
Data "A","B","C","D","E","F","G","H","I","J","K","L","M"
Data "N","O","P","Q","R","S","T","U","V","W","X","Y","Z"
Data "1","2","3","4","5","6","7","8","9","0"
```

2

Ejemplo de impresión de una frase utilizando imágenes en vez de fuentes de letras.

principal que llama a estas funciones.

Una vez definidas las matrices podemos proceder a crear las diferentes funciones.

FUNCIÓN PARA IMPRIMIR UNA FRASE

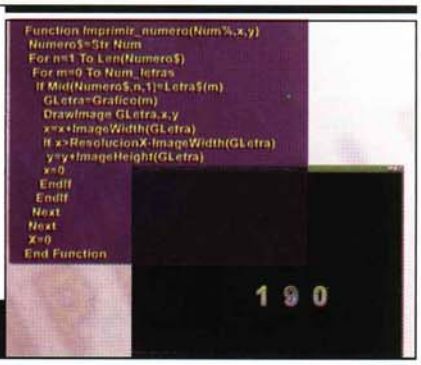
Esta función toma como dato de entrada la frase que queremos imprimir y la posición en la pantalla donde lo hará:

```
Function Imprimir_frase
( Frase$, x#, y#).
```

Ya dentro de la función, convertimos todos los caracteres de la frase entrante en mayúsculas antes de entrar en el bucle de búsqueda, conversión e impresión:

```
Frase$ = Upper ( Frase$ ).
```

Este bucle recorrerá cada uno de los caracteres de la frase y los comparará con los de la matriz de letras. Si coinciden, se imprimirá en pantalla la imagen del carácter correspondiente:



3

Función para imprimir una variable numérica utilizando un alfabeto de diseño.

```
For n = 1 To Len ( Frase$ )
  For m=0 To Num_letras
    If Mid(Frase$,n,1)=Letra$(m)
      GLetra=Grafico(m)
      DrawImage GLetra,x,y
      x=x+ImageWidth(GLetra)
      If x>ResolucionX-ImageWidth(GLetra)
        y=y+ImageHeight(GLetra)
        x=0
      EndIf
    EndIf
  Next
Next
```

Dentro del bucle principal que recorre toda la frase entramos en otro que recorrerá nuestra matriz de letras. Con la instrucción "Mid" aislamos el carácter correspondiente de la frase entrante y lo comparamos con cada letra de nuestra matriz. Cuando coincidan, imprimimos en pantalla su imagen correspondiente controlando, claro está, su posición en la pantalla gráfica (Ver "ejemplo1.bb") .

FUNCIÓN PARA IMPRIMIR UNA VARIABLE NUMÉRICA

Esta función servirá para imprimir cualquier variable numérica (en nuestro caso sólo números enteros) como puede ser un contador de puntuación o de tiempo con nuestro alfabeto de diseño.

Básicamente funciona igual que la función anterior a diferencia que el dato de entrada "Frase$" se sustituye por un valor numérico "Num%". Para poder imprimir los caracteres numéricos correspondientes, es preciso convertir el número entrante en alfanumérico con la función "Str":

```
Function Imprimir_Numero
( Num%, x, y )
    Numero$ = Str Num
```

El resto de la función es exactamente igual (Ver "ejemplo2bb").

FUNCIÓN DE ENTRADA DE DATOS

En esta función se va almacenando en una variable global alfanu-

4

Función para la obtención de datos desde el teclado utilizando el sistema de impresión de caracteres gráficos.

mérica cada carácter introducido desde el teclado con la instrucción "GetKey()". A medida que los caracteres se van introduciendo, se imprimen en pantalla. Saldremos de la función al pulsar la tecla de retorno "Return". En el "ejemplo3.bb" se muestra el modo de utilizar esta función. La variable global que almacenará la frase completa es "Entrada$".

En el "ejemplo4.bb" se muestra este sistema de impresión de caracteres gráficos en un entorno 3D, el cual podría ser cualquier videojuego.

Este tipo de técnica resulta muy útil para dar mayor riqueza gráfica a nuestros juegos. Utilizada adecuadamente, es fácil modificar todos los textos de nuestro juego a diferentes idiomas.

5

Resultado de un ejemplo práctico del sistema de impresión de caracteres gráficos para juegos.

En el próximo número...

... veremos una técnica eficaz para crear y organizar una tabla de récords.

Juegos de estrategia (y IV). juegos divinos y de puzzles

Terminamos la serie dedicada a los juegos de estrategia en tiempo real, hablando de un subgénero muy especial con mezclas de simulación y que ha suministrado títulos de gran espectacularidad, belleza y que, sobre todo, ha proporcionado una forma de jugabilidad inigualable.

Nos referimos a los juegos divinos. Aunque en el número anterior vimos juegos como la serie *RailRoad* o la serie *Theme Park*, que pueden estar incluidos en este subgénero, hemos querido hacer un estudio aparte del trabajo de Peter Molyneux, el padre de los "God-Games". También hablaremos de un género de estrategia que nos ha acompañado desde los primeros tiempos de la historia del videojuego: los juegos de puzzles.

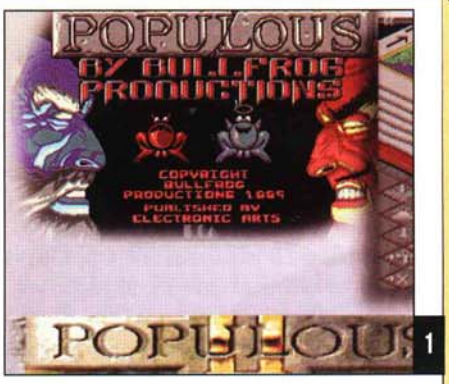
SIMULADORES DE DIOS

Encontramos un género, mezcla de simulación y estrategia, que levantó pasiones desde la publicación por Electronic Arts de *Populous* en 1989; un juego desarrollado por Bullfrog de la mano del gurú del entretenimiento Peter Molyneux. El concepto que mostraba *Populous* era convertir al jugador en una deidad. El cometido principal era hacer más confortable la vida del pueblo para así obtener poderes y destruir a los demás dioses, quienes tenían los mismos objetivos. Como dios, el jugador poseía algunos poderes para destruir como fuego, terremotos, huracanes, etc. Ese mismo año aparecía *Populous: The Promised Lands* con más de lo mismo. La saga se extendió y a lo largo de la década se desarrollaron nuevos títulos como: *Populous II: Trials of the Olympian Gods* (1993) o *Populous 3: The Beginning* (1998), el cual, y siguiendo las líneas tecnológicas del momento, pasa de la isometría a un entorno completamente 3D y donde el jugador se convierte en el dios de un pequeño planeta en el que no sólo controla a sus habitantes sino la Tierra. Un año después, aparece una expansión de *The Beginning* llamada *Undiscovered Worlds* (1999), en la que la historia se basa en la llegada al cielo para restablecer el equilibrio en el mundo de los dioses.

Antes de la publicación de *Populous II*, la casa de Molyneux desarrolla *Dungeon Keeper* (1997), donde el jugador tiene que construir y controlar un mundo bajo tierra aterrorizando a sus habitantes y destruyendo enemigos. La segunda parte se publicó dos años después con mejores gráficos pero con el mismo argumento.

Molyneux deja Bullfrog en 1997 y funda LionHead Studios para crear el God-Game más espectacular y original jamás visto hasta entonces: *Black & White (2001)*

El concepto impuesto para *Black & White* supuso un revulsivo en la industria del videojuego. Su argumento inicial seguía siendo el papel de dios que asume el jugador. En el juego existe una combinación única entre estrategia, construcción de mundos y simulación social con una línea argumental basada en juegos de rol que permite al jugador ser un dios bueno o malo encarnado en una gran criatura. Nuestra criatura es capaz de aprender y de seguir patrones independientes de comportamiento. Posee un gran nivel técnico con aspectos gráficos como: zoom infinito, física y climatología real o luces y sombras dinámicas. Además, el juego está conectado con la vida real; así, por ejemplo, podemos asignar a una criatura para que lea nuestro correo electrónico. Pero sobre todo es su IA



1

La serie Populous, pionera del género God-Games.

2

Black & White revolucionó la forma de diseñar juegos de estrategia.

3

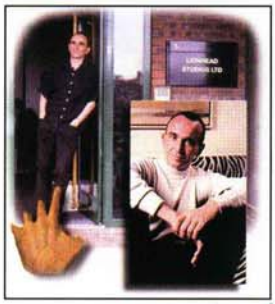
Tras el éxito de Populous, Bullfrog desarrolló Dungeon Keeper, donde el concepto de juego divino se traslada a un mundo subterráneo.

LA BIOGRAFÍA...

PETER MOLYNEUX

Creador del **God-Game**

Considerado como uno de los mejores creadores de juegos de todos los tiempos, Peter Molyneux empezó programando hojas de cálculos. Entró en el mundo de los juegos para ordenador con *Fusion* en 1987, un juego de acción que provocó la creación de la empresa Bullfrog. Con Bullfrog desarrolló un nuevo concepto de jugabilidad que plasmó en *Populous* y en el que ponía al jugador en el papel de un dios. A raíz de este título Molyneux se consagró como unos de los más originales diseñadores de juegos y pronto su empresa creció sin límites. Siguió desarrollando God-Games como *Dungeon Keeper*, *Syndicate* o *Theme Park*, sin olvidar la serie *Populous*. En 1997 deja la compañía Bullfrog después de ser comprada por Electronic Arts y funda LionHead Studios en 1997. Su nuevo equipo se queda perplejo ante la nueva idea que Molyneux tenía entre manos: *Black & White*. Con este título, alcanza la definición absoluta de "juego divino". Y su nueva empresa se convierte en el núcleo de otras compañías afiliadas como: Big Blue Box Studios, Black & White Studios e Intrepid Games, los cuales desarrollan sin parar las nuevas invenciones de este gran Gurú del sector.



Peter Molyneux

lo que sobresale del juego. Los habitantes de *Eden* tienen vida propia y son capaces de simular alegría, tristeza, gratitud o temor además de la posibilidad emprendedora típica del ser humano. No podemos olvidar la interfaz de usuario única en el que el jugador sólo tiene que manejar el puntero del ratón convertido en mano.

Posteriormente a este título aparecieron nuevas partes como *B&W: Criature Isle, B&W: Next Generation* y en curso *Black and White 2*.

JUEGOS DE PUZZLES

Terminamos este capítulo dedicado a los juegos de estrategia con el género más conocido y popular de todos. Los también denominados "estruja cerebros" o juegos de habilidad mental. Prácticamente, los juegos de puzzles han sido el tipo de estrategia que ha dominado todos los tipos de consolas, ya que precisan de pocos recursos gráficos como aceleración 3D o sonido envolvente. Este tipo de juegos era el favorito en las recreativas de los años 80. La sencillez de gráficos necesarios para lograr un título más o menos aceptable produjo una avalancha de producción para todas las consolas del momento ya que prácticamente, los títulos de consolas se apoyaban en juegos de puzzles para reunir a la familia en torno al televisor. La mayoría de estos juegos para consolas han sido versionados para PC o están disponibles a través de emuladores especiales.

En realidad, se podría hablar de un antes y un después del clásico *Tetris* de Alexey Pajitnov (1985). Todos los juegos posteriores a este título están, de alguna manera, influenciados en su sólido concepto. La idea básica de este género es mostrar al jugador numerosos rompecabezas basados en reorganizar una serie de formas y colores en un patrón predefinido. Antes de la salida al mercado del *Tetris*, entre 1976 y 1987 existieron títulos de tipo educacional, como *Othello* o *Tic-Tac-Toe* (1977), basados en juegos populares. Pero los que más sobresalieron en la época fueron: *Maze* (1976), *Qix* (1981), *QBert* (1982) o *Locomotion* (1982).

Al publicarse *Tetris* para ordenadores personales en 1987, todo cambió. El concepto de videojuego de puzzle adquirió la premisa básica de encajar piezas de diferentes colores y formas entre sí. Está basado en la utilización de 7 figuras formadas por 4 pequeños bloques unidos de diferente forma (la palabra "tetris" viene de "tetra", 4). *Tetris* ha sido el juego más versionado de todos y el único que ha aparecido en todas las formas de máquinas de juegos conocidas, desde las recreativas hasta la consola de llavero. Después del *Tetris*, todas las desarrolladoras de equipos lúdicos tenían su propia versión. Algunas de ellas optaron por representar el "mundo tetris" en 3D (*Block Out*, 1989) o simulado (*Klax*, 1989) Pero pocos llegaron desarrollar un estilo personal y original como en el caso de *Ishido* (1990), *Bust-A-Move* (1994), *Panic Bomber* (1994) o *Columns* (1990), este último con una jugabilidad incluso superior, ya que permitía muchas más combinaciones en el juego. También aparecieron nuevos conceptos utilizando otras maneras de usar las formas y colores mediante imágenes y figuras de objetos. Las posibilidades de los ordenadores personales como el PC abrieron las puertas a nuevos conceptos en juegos de puzzles. Entonces surgió un título ingenioso y único: *Lemmings* (1992). El jugador debe controlar a unos hombrecitos para llevarlos sanos y salvos a la salida del nivel por medio de ciertas habilidades. Poseía unos gráficos y animaciones fantásticos para la época. Rompió con el esquema de juegos de habilidad por el del sistema ensayo - error.

Existen infinidad de títulos para este género. En 2D o 3D, con mejores o peores gráficos, todos siguen la premisa de mantener al jugador o jugadores pegados al monitor resolviendo puzzles y desarrollando la agilidad mental sin ningún tipo de violencia.

En el próximo número...

... hablaremos de los juegos de rol.

Cuestionario Videojuegos 13

Preguntas

1. ¿Cómo podemos modificar el tamaño de la vista de una cámara en Blitz3D?
2. ¿Cómo podemos saber con el puntero del ratón la posición de un objeto 3D en Blitz3D?
3. ¿Cómo podemos aislar una colisión para saber con qué elemento de un mismo tipo hemos colisionado?
4. ¿Cómo podemos colocar un sprite en un polígono con la misma orientación que éste?
5. En Milkshape3D, ¿qué operación podemos utilizar para modelar la estructura de una roca?
6. Antes de poder pintar en Deep Paint 3D, ¿qué es lo que debe tener creado el modelo?
7. Si queremos grabar una pista MIDI en Anvil Studio con un teclado externo, ¿qué es lo que debemos preparar primero para que sea posible?
8. ¿Cómo podemos cuantizar una pista en Anvil Studio?
9. Define y lee una base de datos que contenga las letras del alfabeto en Blitz3D.
10. ¿Cómo podemos sustituir una letra de una frase por una imagen en Blitz3D?

Respuestas al cuestionario 12

▷ **1.** "LoadAnimMesh" se utiliza para cargar un modelo con o sin animación. En caso de no contener animación, cargaría todas las partes del modelo si éste no estuviera agrupado. En caso contrario, obtendríamos la secuencia de animaciones completa.

▷ **2.**
```
Modelo_hombre = LoadAnimMesh ("hombre_andando.3ds")
Animate Modelo_hombre, 1
```

▷ **3.**
```
;Definimos la estructura de datos
Type tipo_disparo
    Field Sprite
    Field x,y,z
End Type
; llamamos a la función para crear un disparo desde la nave
Crear_Disparo.tipo_disparo(pivote_nave,sprite_disparo)
  ;Funcion para crear el disparo
Function Crear_Disparo.Tipo_disparo(pivote_nave,entidad_disparo)
    Disparo.tipo_disparo = New tipo_disparo
    Disparo\Sprite= CopyEntity (entidad_disparo, pivote_nave)
    Disparo\x= EntityX(pivote_nave)
    Disparo\z= EntityZ(pivote_nave)
    Disparo\y= EntityY(pivote_nave)
End Function
```

▷ **4.** Llamando a la función de actualización para cada disparo con un bucle:
```
For Disparo.tipo_disparo= Each tipo_disparo
    Actualizar_disparo(disparo)
Next
```

▷ **5.** Activando la casilla "IK Chain Terminator" en la ventana de opciones *Tool Option* en *Joint*.

▷ **6.** Modificando el "framerate" manualmente. Para ello, elegimos la opción *Custom* en *Set framerate* del menú *Animation*.

▷ **7.** En Anvil Studio se pueden crear pistas MIDI (instrumentos), de ritmos y pistas de audio.

▷ **8.** Los eventos MIDI en Anvil Studio se editan en la sección *Compose*.

▷ **9.** Utilizando planos, por medio de una primitiva cúbica o utilizando mapeado cúbico.

▷ **10.** En primer lugar debemos preparar las texturas en un programa de dibujo. Seguidamente, modelamos un cubo lado a lado. Cada lado lo formamos uniendo vértices y triángulos en una superficie y que posteriormente texturizamos con un *brush*.

Contenido CD-ROM 13

▶ AUDIO

■ AnalogX Scratch
Pequeño programa gratuito para realizar scratches y luego exportarlos a un fichero .wav.

■ Audacity
Completo editor de audio con el que grabar, mezclar y editar pistas de audio con calidad CD.

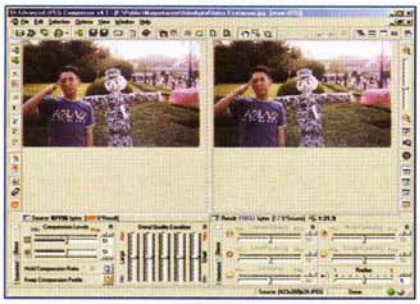
■ Groovebox
Maravilloso programita hecho en Flash con el que podrás realizar un montón de mezclas en vivo.

■ JV Percussion Generator Light 4
Genera percusiones a partir de simples palabras. Muy divertido.

■ MbooM
Secuenciador que además permite el tratamiento de melodías.

▶ DISEÑO 2D

■ Advanced JPEG Compressor
Herramienta que nos asiste en la tarea de convertir y comprimir imágenes en formato JPG, para el uso en la web.



■ DiagramStudio 2.2
Con este potente programa podrás crear diagramas para poner en orden tus ideas y proyectos.

■ FireGraphic XP
Solución completa para visualizar, ordenar e imprimir imágenes con rapidez.

■ Photolightning
Sencillo programa para optimizar y realizar cambios rápidos en nuestros archivos gráficos.

■ PhotoMeister 1.22
Con esta utilidad podrás crear completos álbumes con todas tus fotos e ilustraciones.

■ Tess 1.41
Usando esta herramienta lograrás dibujar formas geométricas en pocos minutos.

▶ DISEÑO 3D

■ D Sculptor 2.0
Editor de objetos 3D muy completo y que produce resultados de una gran profesionalidad.

■ Poser 4
Uno de los programas de diseño 3D más utilizados y potentes con el que podrás modelar espectaculares personajes.

■ RayArtTan 1.0
Render para animaciones en tres dimensiones muy fácil de usar.

■ SolidWorks Property Propa-Gator
Maneja eficientemente tus archivos de SolidWorks.

▶ PROGRAMACIÓN

■ SuperEditor 2.01
Simplifica tus tareas de programación con este potente editor con el que verás el código mucho más claro.

■ Defect Agent 3.0
Con esta práctica utilidad podrás ir tomando nota de todos los fallos en el desarollo.

■ EasyLicenser License Manager
Protege el software que crees con esta herramienta de control de licencias.

■ Emeditor
Pequeño editor con soporte para múltiples lenguajes de programación.

■ InstallConstruct 5.3
Crea instaladores, agentes de ayuda y desinstaladores para tus programas.

■ Video2000
Evalúa la calidad de tus imágenes y del juego para ver si éste se visualiza correctemente.

▶ JUEGOS

■ Block Out 1.58
Famoso juego que no es sino un peculiar Tetris en tres dimensiones.

■ Bust-A-Move 4
Simpático puzzle en el que debes explotar todas las burbujas a tiempo.

■ Populous The Beginning
Demo del estupendo juego en el que adquirías el poder de ser un dios.

■ Lemmings 3D
Versión en tres dimensiones de los Lemmings, que tantos adeptos sigue teniendo.

■ Tetristation 1.0
Juega al clásico Tetris en esta nueva versión, que sin duda te enganchará.

■ Zone of Fighters
Como todas las semanas, nuestro juego.

▶ VÍDEO

■ Joiner 1.1
Aplicación para unir y sincronizar vídeo y audio muy fácilmente.

■ M2 Edit Pro 5.0
Editor avanzado de MPEG-2 con el que podrás realizar auténticas virguerías.

■ WinMPG Video Convert 1.50
Convierte tus archivos de vídeo en distintos formatos usando este programa.

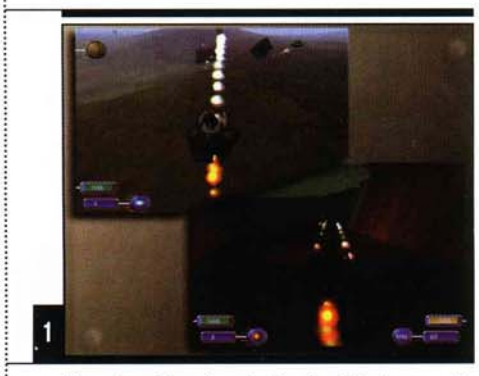
▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.

Efectos especiales. Sistema de partículas

En esta entrega vamos a desarrollar una de las partes del videojuego que proporciona más espectacularidad a la acción y ayuda a crear un entorno más real.

Nos referimos al sistema de partículas que gestiona todos los efectos especiales de fuego, humo, chorros de luz y otros de carácter climático, como la nieve o la lluvia. Los sistemas de partículas suelen ser bastante complejos de implementar, sobre todo si se quieren buscar efectos realistas. Básicamente, los sistemas de partículas consisten en el manejo independiente de entidades, normalmente sprites, asignando a cada una de ellas unas propiedades físicas. Cada una de estas propiedades determina el comportamiento de las partículas que, unidas, forman un determinado efecto. Dependiendo del número de estas propiedades podemos obtener un sistema más o menos complejo y real. Evidentemente, a más variables mayores posibilidades.

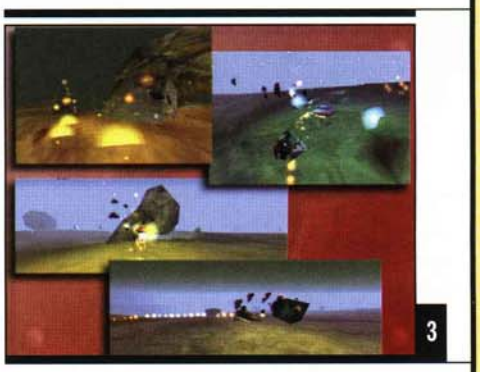


Ejemplos del emisor de tipo 1, utilizado para el humo de los disparos.

Ejemplos del emisor de tipo 2, utilizado para simular los motores de la bionave y los ovnis.

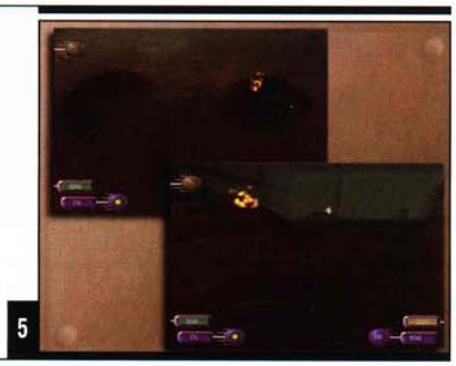
¿CÓMO SE FABRICA UN EMISOR DE PARTÍCULAS?

Hay multitud de maneras de construir un sistema de partículas, como es normal dependiendo siempre de nuestras necesidades. Lo más estándar y útil es implementar un sistema general, a través de funciones, que admita valores para modificar las variables que gobiernan a las partículas. De esta forma, sólo tenemos que llamar a la función con unos valores determinados para conseguir diferentes efectos. En "Zone of Fighters" trabajaremos con una función que crea lo que se denomina "emisores". Los emisores son zonas cuadradas o rectangulares, de tamaño variable, que pueden estar situadas en cualquier lugar y donde colocamos las partículas para que evolucionen. El número de ellas determina la densidad del efecto. Por ejemplo, si queremos simular la nieve al caer, obtendremos mejores resultados si creamos mucha cantidad de partículas, ya que cada una de ellas representa a un copo. Aunque cada partícula puede ser también una entidad en 3D como cubos o esferas, es más normal el uso de sprites texturizados y siempre orientados hacia la cámara. Y ¿por qué sprites? Sencillamente, porque el uso de emisores de partículas conlleva la creación de muchas entidades para obtener un resultado más o menos decente y eso supone muchos polígonos y por consiguiente un mayor consumo de rendimiento. Si, por ejemplo, usamos cubos, necesitaríamos 12 polígonos por cada uno de ellos; así que para crear un emisor de 100 partículas se tendrían que dibujar 1200 polígonos. Como sabemos, un sprite es un objeto plano formado por sólo dos polígonos. Si lo orientamos hacia la cámara y le aplicamos una textura adecuada se pueden obtener resultados fantásticos y solo necesitaríamos 2 polígonos por partícula. La idea principal es



Ejemplos del emisor de tipo 3, utilizado en todas las explosiones.

Dos capturas del emisor de tipo 5, especializado en generar efectos climatológicos como la nieve.



5

El emisor de tipo 2 también se puede utilizar para generar el fuego de los volcanes.

crear un número inicial de partículas por medio de sprites. Luego, durante el juego, vamos modificando, actualizando y destruyendo estos sprites. Una vez que se llama a la función, ésta crea un emisor en donde evolucionan las partículas siguiendo los valores de las variables de entrada a través de ciertas operaciones matemáticas.

DEFINIENDO LA ESTRUCTURA DE DATOS

Antes de crear las partículas debemos definir cada una de las variables que intervendrán en su evolución en el módulo "definiciones.bb". Como vamos a utilizar un gran número de entidades iguales y que tenemos que actualizar durante el juego, lo mejor es utilizar una estructura de datos (Ver Código 1).

Bien, en primer lugar vamos a dividir esta estructura para que se pueda comprender mejor. Por un lado, tenemos las variables que utilizaremos para el emisor y por otro las variables para cada partícula.

CREANDO LAS PARTÍCULAS PARA CADA TIPO DE EMISOR

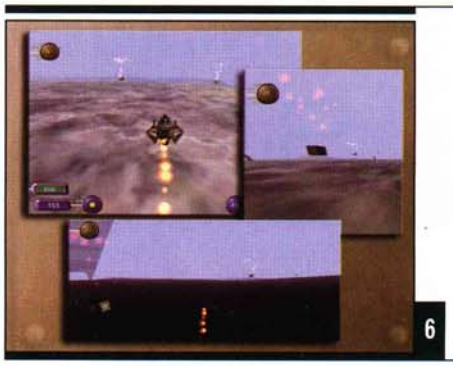
En el juego, utilizamos 6 emisores diferentes:

- **Emisor 1:** Se utiliza para el humo de los disparos de misiles y bombas de minifusión.
- **Emisor 2 y 3:** Se utilizan en las explosiones y para el motor de la bionave y los ovnis.
- **Emisor 4:** Se utiliza para el chorro de luz que emite cada bono.
- **Emisor 5:** Se utiliza para generar la nieve.
- **Emisor 6:** Se utiliza para generar la lluvia.

Debemos definir el mismo tipo de estructura y crear un número determinado de sprites para cada tipo de emisor. La creación de las partículas (sprites) para cada emisor la realizamos en la función "crear_particulas()" en el módulo "funcpantaudio.bb". Utilizamos un bucle desde 1 hasta el número total de partículas de cada emisor a partir de la variable global "Num_particulas", por ejemplo, para el emisor 1 tenemos:

```
For x = 1 To Num_particulas*3 ;
Num_Particulas=100 * 3 =
300 partículas para el emisor 1
  b.emisor = New emisor
  b\Entidad = CopyEntity(particula)
  b\Alfa# = Rnd(.2,.8)
  EntityAlpha(b\Entidad,b\Alfa#)
  EntityBlend b\Entidad,3
Next
```

Aquí, solo creamos el sprite ("particula") que asignamos a "Entidad" y, seguidamente, definimos su transparencia inicial. Los demás valores se irán creando y actualizando en la función de actualización del emisor.



6

Los emisores de tipo 4 se utilizan sólo para el chorro de luz que sale de los bonos.

Podemos tener varios emisores del mismo tipo; es decir, si en el juego se producen al mismo tiempo cinco explosiones, tendremos a la misma vez cinco emisores del tipo 2 o 3. Ahora bien, si hemos creado 300 partículas para el emisor del tipo 2, en todas las explosiones que utilicen este emisor se repartirían estas partículas. Del mismo modo sucede con todos los demás tipos.

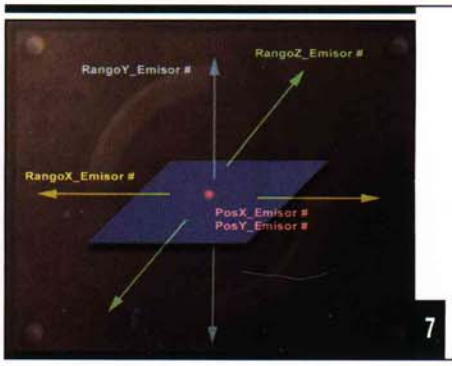
GESTIONANDO LAS PARTÍCULAS DE CADA EMISOR

Ha llegado el momento de construir el núcleo de nuestro sistema de partículas. Para cada tipo de emisor utilizamos un procedimiento diferente pero con el mismo funcionamiento. Así que tomaremos como ejemplo la del emisor de tipo 1 (Ver Fig.7).

Para crear un emisor desde cualquier punto del programa llamamos a la función con los valo-

Código 1. Usamos una estructura de datos

```
Type emisor
  ; Emisor
  Field X_Emisor#,Y_Emisor#,Z_Emisor#   ; Posición del emisor
  Field RangoX#,RangoY#,RangoZ#         ; Tamaño del emisor
  ; Particula
  Field Entidad                         ; Entidad Sprite que forma la partícula
  Field Vida                            ; Vida de la partícula una vez emitida
  Field Alfa#                           ; Transparencia de la partícula
  Field Angulo_Rotacion#                ; Angulo de rotación de cada partícula
  Field Xvel#,Yvel#,Zvel#               ;  Velocidad inicial de las partículas
End Type
```



7

Esquema de un emisor y sus parámetros de posición y tamaño.

res correspondientes. Por ejemplo, vamos a asignar unos valores previos y llamamos a la función pasando esos valores (Ver Código 2).

Las variables *PosX_Emisor, PosY_Emisor* y *PosZ_Emisor* corresponden a la posición del emisor de partículas. *RangoX_Emisor, RangoY_Emisor* y *RangoZ_Emisor*, definen el tamaño del emisor. *VeloX_Particula, VeloY_Particula* y *VeloZ_Particula*, asignan la velocidad inicial de las partículas. *Vida_Particula* asigna el tiempo que estará la partícula en acción. *Tamaño_Particula* y *Peso_Particula* definen el tamaño inicial que tendrá el sprite de la partícula y su peso. Y *Dir_Viento* y *Vel_Viento* asignan la dirección y velocidad que tendrá el viento.

Cada función de control de los emisores la incluimos en el módulo "funcpantaudio.bb" y su descripción la vemos en el Código 3.

En primer lugar, preguntamos si la partícula ha muerto a través de su variable "Vida". En caso afirmativo (*b\Vida < 0*), procedemos a inicializarla con los valores iniciales. Recordemos que no podemos destruirla, ya que sólo la creamos una vez en el programa. Por lo tanto, sólo debemos inicializarla en posición, velocidad inicial, tamaño y tiempo de vida.

Si la partícula no ha llegado al final de su vida la actualizamos según los siguientes procesos:

A)

```
b\X_Emisor#=b\X_Emisor#+b\Xvel#
b\Xvel#=b\Xvel#-.001+Dir_Viento#
```

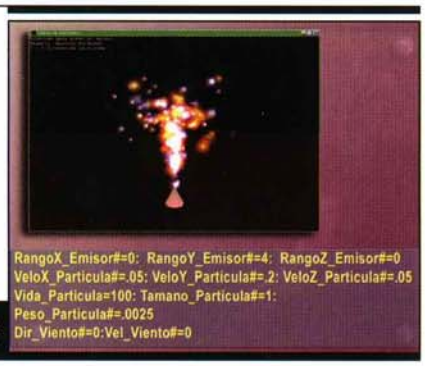
Incrementamos la posición inicial de la partícula en la dirección del eje X según la velocidad indicada en "Xvel". Además, reducimos este incremento según la dirección del viento (*.001+Dir_Viento#*) para tener en cuenta la resistencia que éste ofrece.

Código 2. Funciones de control de los emisores

```
PosX_Emisor# = 100 :  PosY_Emisor # = 100 :PosZ_Emisor #= 100
RangoX_Emisor# = 50 : RangoY_Emisor# = 10: RangoZ_Emiso#r = 50
VeloX_Particula# = 0 : VeloY_Particula# = 0: VeloZ_Particula# = 0
Vida_Particula = 300 : Tamaño_Particula# = 5 : Peso_Particula# = 1
Dir_Viento# = 2 :  Vel_Viento# = .5
emisorparticulas (PosX_Emisor#, PosY_Emisor#,PosZ_Emisor#,
                  RangoX_Emisor#, RangoY_Emisor#, RangoZ_Emisor#,
                  VeloX_Particula#, VeloY_Particula#, VeloZ_Particula#,
                  Vida_Particula, Tamaño_Particula#, Peso_Particula#,
                  Dir_Viento#, Vel_Viento#)
```

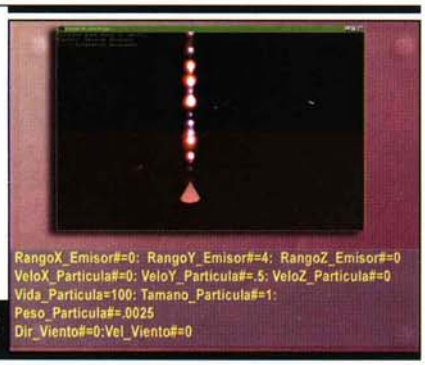
Código 3. xxxxxxxxx

```
Function emisorparticulas  (PosX_Emisor#,PosY_Emisor#,PosZ_Emisor#,
                            RangoX_Emisor#,RangoY_Emisor#,RangoZ_Emisor#,
                            VeloX_Particula#, VeloY_Particula#, VeloZ_Particula#,
                            Vida_Particula, Tamaño_Particula#, Peso_Particula#,
                            Dir_Viento#, Vel_Viento#)
 For b.emisor = Each emisor
  If b\Vida<0
   b\X_Emisor#=Rnd(PosX_Emisor, PosX_Emisor +RangoX_Emisor#)
   b\Y_Emisor#=Rnd(PosY_Emisor, PosY_Emisor +RangoY_Emisor#)
   b\Z_Emisor#=Rnd(PosZ_Emisor, PosZ_Emisor +RangoZ_Emisor#)
   b\Xvel#=Rnd(-VeloX_Particula#,VeloX_Particula#)
   b\Yvel#=VeloY_Particula#
   b\Zvel#=Rnd(-VeloX_Particula#,VeloZ_Particula#)
   Escala#=Rnd(.1,Tamaño_Particula)
   ScaleSprite  (b\Entidad,Escala#,Escala#)
   b\Vida=Rnd(Vida_Particula-40,Vida_Particula+40)
  EndIf
  b\X_Emisor#=b\X_Emisor#+b\Xvel#
  b\Xvel#=b\Xvel#-.001+Dir_Viento#        ; Resistencia al viento
  b\Y_Emisor#=b\Y_Emisor#+b\YVel#
  b\Yvel#=b\Yvel#-Peso_Particula#    ; Gravedad
  b\Z_Emisor#=b\Z_Emisor#+b\Zvel#
  b\Zvel#=b\Zvel#-.001+Vel_Viento#         ; Aumenta la velocidad por el viento
  b\Vida = b\Vida - 1
  PositionEntity b\Entidad,b\X_Emisor#,b\Y_Emisor#,b\Z_Emisor#
  b\Angulo_Rotacion#=b\ Angulo_Rotacion #+2
  If b\ Angulo_Rotacion #>360 b\ Angulo_Rotacion #=1
  RotateSprite b\Entidad,b\ Angulo_Rotacion #
 Next
End Function
```



8

Valores necesarios para obtener un chorro de partículas estándar utilizando el programa de testeo "particulas.bb".



9

Un ejemplo de hilo de luz ideal para el humo de los misiles o la combustión de un motor.

B)

```
b\Y_Emisor#=b\Y_Emisor#+b\YVel#
b\Yvel#=b\Yvel#-Peso_Particula#
```

Aquí, determinamos el desplazamiento vertical de cada partícula incrementando la variable que controla el eje Y "Y_Emisor" según la velocidad dada por "Yvel". También añadimos un factor de gravedad (*.01+Peso_Particula#*) que dependerá del peso que le asignemos a la partícula.

10

Fuente de luz expansiva, un efecto muy llamativo que puede ser utilizado para generar explosiones de estrellas.

C)

```
b\Z_Emisor#=b\Z_Emisor#+b\Zvel#
b\Zvel#=b\Zvel#-.001+Vel_Viento#
```

El siguiente paso es determinar el desplazamiento en el eje Z ("Z_Emisor") según el valor de la velocidad inicial de la partícula almacenada en "Zvel". Añadimos también un factor extra como la velocidad del viento ("Vel_Viento#")

D)

```
b\Vida = b\Vida - 1
```

Decrementamos el valor de la vida de la partícula para dar paso a su siguiente estado.

E)

```
PositionEntity b\Entidad,
b\X_Emisor#,b\Y_Emisor#,b\Z_Emisor#
b\Angulo_Rotacion#=
b\ Angulo_Rotacion #+2
If b\ Angulo_Rotacion #>360
b\ Angulo_Rotacion #=1
RotateSprite b\Entidad,
b\ Angulo_Rotacion #
```

Y por último, dibujamos el sprite y lo rotamos según el ángulo definido por "Angulo_Rotacion#".

Como se puede observar no es realmente un procedimiento complicado. Resulta sencillo y además es bastante eficaz.

CREANDO EFECTOS ESPECIALES

Una vez que hemos completado nuestro sistema de partículas, es el momento de disfrutarlo. Para obtener diferentes efectos basta con pasar a la función distintos valores. Para comprobar el funcionamiento de la función vamos a realizar pruebas con un pequeño programa de testeo incluido en "Extras" del CD, llamado "Particulas.bb". En él utilizamos exactamente el mismo sistema de partículas que acabamos de explicar, el que aparece en el juego. La única diferencia son las medidas de los valores, ya que creamos un entorno 3D estándar. Así que los valores que debemos colocar deben ser mil veces menor es que los utilizados en el juego.

11

Ejemplo de emisor en llamas. Es necesario aumentar el tamaño de las partículas y mantener una distancia cercana entre ellos para conseguir este efecto.

12

Ejemplo de llama de fuego inclinada por el efecto del viento.

Los diferentes efectos que podemos crear dependerán de los valores que asignemos a los parámetros de la función. Es fácil utilizar el ejemplo que se acompaña para hacer pruebas con distintos valores. Apuntando estos valores podemos tener una buena librería de efectos que podemos aplicar en nuestros juegos.

El gestor de partículas mantiene una estructura abierta. Por lo tanto, es posible la ampliación de nuevas posibilidades, como por ejemplo, tener en cuenta la elasticidad o peso de cada partícula, necesario para implementar el bote con el suelo u otras entidades.

En el próximo número...

... daremos vida al terreno de juego, colocando todo el decorado y seres vivos.

Fabricando los elementos del juego (V)

En el número anterior, terminamos de modelar los elementos que componen el decorado y ahora queda texturizarlos.

En esta ocasión, necesitaremos usar varios programas para llevar a cabo esta labor. En primer lugar, fabricaremos las plantillas necesarias que nos sirvan de guía en Paint Shop Pro. Para ello, utilizaremos el programa LithUnwrap. También, usaremos la aplicación de diseño de entornos Bryce (cualquier versión es válida) que nos ayudará a obtener texturas más realistas.

CREANDO LA PLANTILLA DE LOS ALMACENES

Antes de entrar en Paint Shop Pro para dibujar las texturas, debemos tener una plantilla o template del mapeado de los almacenes. Esta plantilla nos ayudará a tener una referencia de por dónde dibujar y así conseguir correctamente cubrir el modelo. Ejecutamos LithUnwrap y cargamos el modelo del primer almacén "almacen1.3ds" (en "Extras" del CD se pueden hallar todos los modelos de esta entrega). Antes de nada, debemos crear el mapeado UV.

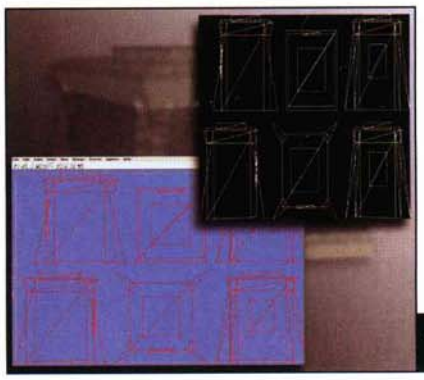
Seleccionamos todos los vértices con la opción *Select All / Select*. A continuación, elegimos la opción *Box* en *Tools/UV Mapping*. Y en la ventana flotante de diálogo *Box Mapping* dejamos seleccionado solo *Apply spacing*. Obtendremos una plantilla con la vista de las seis partes del almacén. Seleccionamos el tamaño que tendrá la imagen de la plantilla. En *Preferences* del menú *Options* nos situamos en la pestaña *Template* y en *Bitmap dimensions* seleccionamos un tamaño de 512 x 512. Para salvar la plantilla, elegimos la opción *Save* en *File/Template*. El programa nos volverá a preguntar las dimensiones de la imagen, que observamos que es 512 x 512. Para el siguiente almacén ("almacén2.3ds") realizamos las mismas operaciones (Ver Fig. 1).

DIBUJANDO LAS TEXTURAS DE LOS ALMACENES

Antes de pintar el aspecto de nuestro primer almacén precisamos crear cada una de las texturas que lo compondrá. Existen multitud de librerías gratuitas de imágenes para utilizar en nuestras texturas. Sin embargo, la realización manual de texturas desde cero es tarea obligada para tener un total control de nuestros diseños.

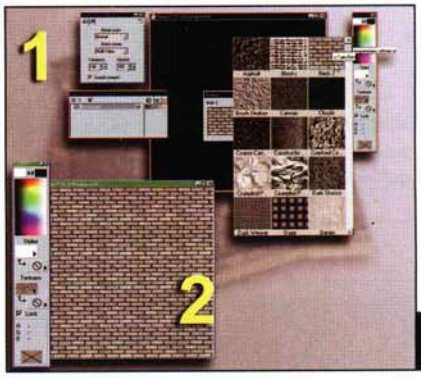
TEXTURA PARA LAS PAREDES

Vamos a empezar creando la textura de ladrillos que forman las paredes. Posteriormente, este dibujo se ajustará a la plantilla del modelo como si de un mosaico se tratara para conformar la textura final. En el aspecto que diseñamos para los almacenes, elegimos unas paredes creadas con ladrillos y con una línea de piedras en la base y bordes del techo. Empezamos, creando las paredes de ladrillo con Paint Shop. Ejecutamos el programa y seleccionamos un documento nuevo de 1024 x 1024 con color negro de fondo. Vamos a rellenar esta ima-



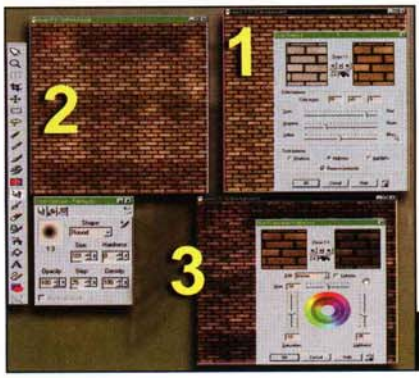
1

Imagen del template (plantilla) creado por LithUnwrap con las seis caras del almacén.



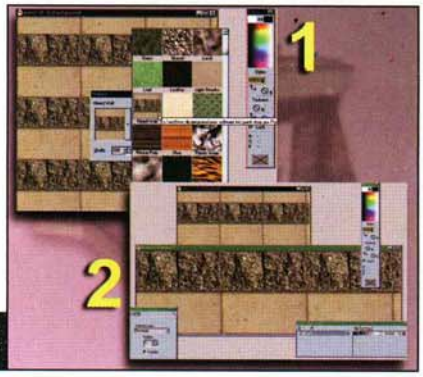
2

Aprovechemos la librería de patrones y texturas de Paint Shop Pro para obtener una pared de ladrillos.



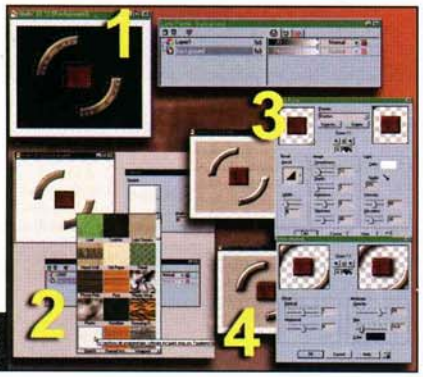
3

Para personalizar nuestra pared se pueden aplicar efectos de iluminación con la herramienta de retoque.



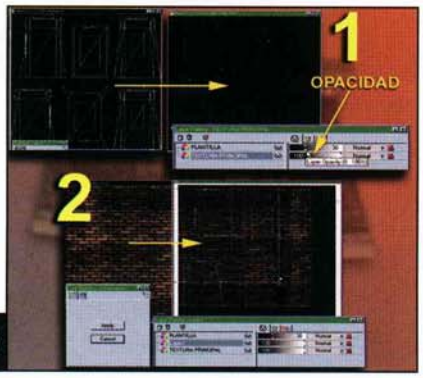
4

Para los bordes de piedra utilizaremos de nuevo la librería de patrones.



5

Debemos aislar el logotipo del fondo para poder aplicar los efectos 3D que le conferirá volumen.



6

Para realizar la textura final utilizaremos la plantilla en otra capa semitransparente como guía. Esta técnica es parecida a las utilizadas en los dibujos animados convencionales.

gen con una textura estándar de la librería de Paint Shop Pro. En la paleta de herramientas *Colors* nos situamos en *Styles* y elegimos un color sólido blanco para la tinta y anulamos el fondo (null). Luego, nos situamos en *Textures* y elegimos para la tinta la textura *Brick 2*de la librería de Paint Shop Pro.
Seguidamente, rellenamos la imagen con la herramienta de relleno *Flood Fill* (tecla F) (Ver Fig. 2).

El siguiente paso será dar un aspecto más real a los ladrillos aplicando algunas funciones de retoque fotográfico. En primer lugar, vamos a cambiar la tonalidad de la imagen mediante la función de balance de color *Color Balance* en el menú *Colors*. Elegimos unos valores para los tonos medios de rojo, verde y azul de 70, -45 y -5. Seguidamente, realizamos algunos retoques en la imagen con *Retouch* del panel de herramientas (Ver Fig. 3 / 1).

En la ventana de opciones de esta herramienta *Tool Options - Retouch*, seleccionamos un tamaño (Size) máximo (255) y una forma cuadrada para el pincel de retoque. Vamos a oscurecer algunas partes de la imagen y aclarar otras con los modos de retoque *Darken RGB* y *Ligthen RGB*. Para eliminar la uniformidad de la textura vamos también a aplicar algunos retoques diferentes como un poco de iluminación con *Lightness up* o un cambio del tono con *Hue up*. Estos retoques los haremos siempre en determinadas partes de la imagen (Ver Fig. 3 / 2). Para finalizar, cambiamos los valores de tono, saturación y brillo en *Colors/ Hue/Saturation/Lightness* a Hue = -10, Saturation = -10 y Lightness = -25. Salvamos la imagen como "pared.bmp" y pasamos a desarrollar la textura de piedra que colocaremos en la base y de separación entre las paredes y la techumbre (Ver Fig. 3 / 3).

TEXTURA PARA LOS BORDES DE PIEDRA Y LOGOTIPO

Borramos la imagen con "Ctrl" + "A" y "Suprimir". En la paleta *Colors* dejamos vacía *Textures* y en *Styles* seleccionamos el patrón (*Pattern*) *Mixed Wall* de la librería. Completamos el proceso rellenando la imagen (Ver Fig. 4 / 1).

Con la herramienta de selección elegimos la porción que se muestra en la figura 4 y la copiamos en una nueva imagen ("Ctrl"+ "C" y "Ctrl"+ "V"). Salvamos esta imagen como "borde.bmp" (Ver Fig. 4 / 2). Terminamos los elementos de la pared preparando la placa con el logotipo del juego. Para ello, cargamos la imagen del logotipo del juego que diseñamos en el número 5. Si no se dispone de ella, en el CD se puede encontrar con el nombre "titulo.bmp". Una vez cargada, aislamos el logo del fondo negro con la herramienta de selección *varita mágica* ("Magic Wand"). Hacemos clic en el fondo negro de la imagen e invertimos la selección en *Selections/Invert* (Ver Fig. 5 / 1).

Posteriormente, hacemos una copia en otra capa con "Ctrl"+ "C" y "Ctrl"+ "L". En la paleta de capas (*Layer Palette*) seleccionamos el fondo y pulsamos la tecla "suprimir" para borrar el logo del fondo. Rellenamos la capa del fondo con el estilo (*Styles*) *Spatch* de la librería con un tamaño (*Scale*) de 50 (Ver Fig. 5 / 2). Lo oscurecemos un poco con la opción *Colors/ Brightness/Contrast* y pasamos a la capa del logo para aplicar un efecto *Inner Bevel* en *Effets/3D Effects* (Ver Fig. 5 / 3) y un sombra con bastante "Blur" con el efecto *Drop Shadow* (Ver Fig. 5 / 4). Salvamos con el nombre "logo.bmp".

El techo lo dibujamos en la textura principal. Así que ya estamos preparados para realizarla utilizando la plantilla que creamos antes.

TEXTURA PRINCIPAL

El paso final es crear la textura definitiva que irá en el almacén utilizando la plantilla y las demás texturas que hemos dibujado. Creamos un documento nuevo de 512 x 512 y fondo negro. Luego, cargamos la plantilla que salvamos con LithUnwrap ("almacen1.bmp"). Debemos ahora crear una nueva capa en la imagen nueva con la plantilla. Para ello, desplazamos con el ratón la capa del fondo de la imagen de la plantilla hasta el nuevo documento. El siguiente paso es variar la transparencia de la capa de la plantilla para poder ver lo que pintamos en la capa del fondo. Este proceso es similar al utilizado para realizar, de manera clásica, los dibujos animados. Para realizar este proceso cambiamos la opacidad de la capa más o menos al 30 % en el modo normal (Ver Fig. 6 / 1).

Ahora solo tenemos que ir rellenando las siluetas con imágenes para tener la textura definitiva. Pasemos a dibujar las paredes. Cargamos la textura de la pared que fabricamos anteriormente y la pasamos a la imagen que estamos trabajando. Observamos cómo resulta de mayor tamaño que su anfitrión, así que tenemos que escalarla con la herramienta de deformación (*Deformation*) hasta que cubra la pared de la primera vista del almacén en la plantilla (incluido los polares de las esquinas) (Ver Fig. 6 / 2). Esta misma capa la copiamos con "Ctrl"+ "C" y "Ctrl"+ "L" para cubrir el resto de vistas. Para mover las capas copiadas, seleccionamos la herramienta de deformación, movemos y ajustamos (Ver Fig. 7).

Sigamos ahora, situando la textura de piedra en la base y en la parte alta. De igual forma que la anterior, cargamos la textura "borde.bmp" y la copiamos a la imagen principal. Seguidamente, escalamos y ajustamos a la base de la primera vista (Ver Fig. 8 / 1).

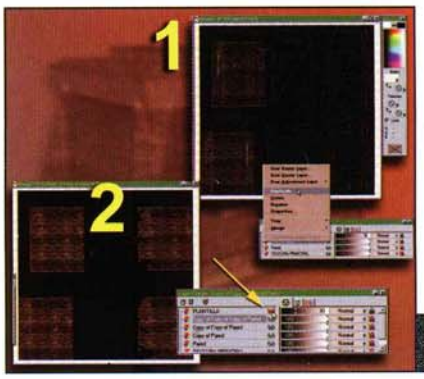
Una vez realizado este proceso, copiamos la capa y la ajustamos a otra vista y así con todas las demás. Hacemos el mismo proceso para la parte superior del edificio (Ver Fig. 8 / 2). Podemos ver el resultado ocultando la plantilla pulsando en las gafas situada al lado del nombre de la capa. Para las vistas del techo y la base utilizaremos un patrón de la librería del programa.

Nos situamos en la capa del fondo y seleccionamos con la herramienta de selección rectangular la vista del techo (la segunda). En *Styles* elegimos el patrón *Pine* y lo escalamos al 99% antes de aplicarlo. Aplicamos el patrón sobre la selección con la herramienta de relleno. Hacemos lo mismo con la vista que nos queda de la base (incluido los pilares) (Ver Fig. 9 / 1).

Ya solo nos queda colocar el logotipo del juego en el espacio reservado en la plantilla. Cargamos la imagen anterior del logo y la pasamos, como es habitual, a la imagen principal (Ver Fig. 9 / 2). Seguidamente, lo ajustamos al cuadrado central de las vistas tercera y sexta con la herramienta de deformación. No hay que preocuparse si la imagen queda cambiada de tamaño (Ver Fig. 9 / 3). Eso es debido a que la textura tiene forma cuadrada. Luego desde programación se ajusta automáticamente siguiendo el mapeado UV.

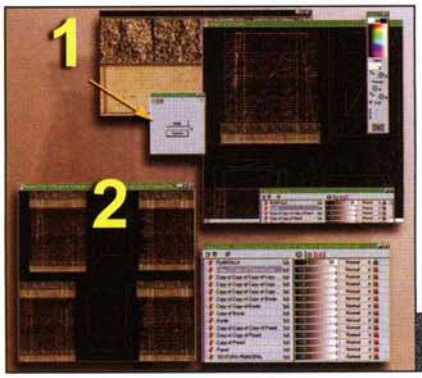
RETOCANDO LA TEXTURA

Si observamos en LithUnwrap el resultado de la textura aplicada al modelo, diremos que está correcto. Sin embargo, podemos lograr un nivel aún más detallista realizando algunos retoques extras. Si miramos con más detenimiento comprobaremos que falta algo en la textura. Realmente, da la sensación que el edificio



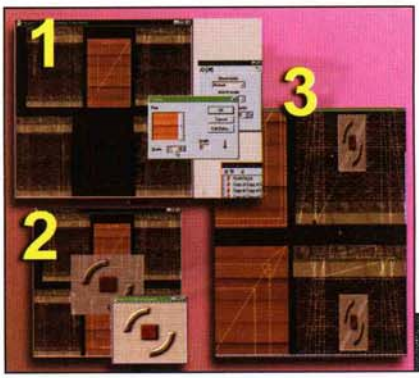
7

Después de ajustar la imagen de la pared a la primera vista de la plantilla, duplicaremos la capa y con la herramienta de transformación dibujaremos las otras vistas.



8

Al igual que hicimos con las paredes, es necesario escalar la imagen del borde de piedra para ajustarlo a la plantilla.

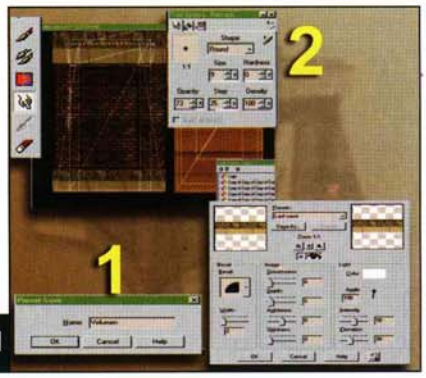


9

Después de rellenar las vistas del techo y la base, debemos colocar la imagen del logotipo del juego.

10

Aunque la textura parece finalizada, todavía es necesario realizar algunos retoques para crear volumen.



11

Para realzar los volúmenes siempre es conveniente aplicar ciertos tipos de efectos de luz.

12

Resultado final de nuestra textura y cómo queda aplicada al modelo.

tiene pegadas las texturas y no que forma parte del material que lo constituye. Debemos entonces aplicar ciertos efectos de iluminación para enfatizar el volumen de las formas. Así que volvamos a Paint Shop Pro para terminar el trabajo.

En primer lugar, apliquemos un efecto de relieve a las capas que conforman los bordes de piedra y el logotipo para dar la sensación de volumen. Seleccionamos la capa del borde inferior de la primera vista y elegimos el efecto 3D *Inner Bevel* en *Effects/3D Effects*. A continuación, elegimos el *Presets* custom y colocamos los siguientes valores:

- En **Bevel:** elegimos la segunda figura y un tamaño ("Width") de 2.
- En **Image:** *Smoothness* 0, *Depth* 5, *Ambience* 0 y *Shininess* 0.
- Un ángulo de 190 en **Angle**, una intensidad (*Intensity*) de 50 y una elevación (*Elevation*) de 30. Salvamos el preset en *Save as...* con el nombre "Volumen" y nos servirá para todas los demás bordes (Ver Fig. 11 / 1).

Una vez aplicado a todos los bordes el mismo efecto, pasemos a definir un poco de sombreado. Vamos a perfilar con la herramienta de retoque en el modo *Darken RGB* todos los bordes que marcan el volumen en la plantilla. Para realizar esta operación es conveniente acercar la imagen con el zoom al menos el doble para poder pintar con detalle (botón izquierdo del ratón aumenta zoom y el derecho disminuye). Pasemos a la primera vista para pintar la pared. Situémonos en la capa correspondiente y seleccionemos la herramienta de retoque. Ajustémosla a un tamaño de 5 y una opacidad (*Opacity*) del 75%. A continuación, pintamos todos los bordes con cuidado siguiendo las líneas marcadas por la plantilla (Ver Fig. 11 / 2). Haremos lo mismo en todas las vistas. El resultado final dependerá de las dotes artísticas de cada uno. Solamente hay que tener en cuenta cómo afectaría la luz al almacén en la vida real. Ahora sí podemos decir que la textura está terminada.

El segundo almacén, aunque tenga diferente forma, tiene el mismo aspecto que el primero. Por lo tanto, utiliza las mismas texturas. Lo único que tenemos que hacer es crear la plantilla y ajustar en Paint Shop Pro las texturas que tenemos de la pared y los bordes de piedra a la nueva plantilla. La única diferencia la encontramos en el techo y la base, los cuales tienen forma pentagonal. Podemos resolver esto de dos maneras. Una primera, seleccionando la vista completa con un rectángulo y rellenar con un patrón, como hasta ahora. Y una segunda, seleccionando con la herramienta de selección *Freehand* en el modo *Point to Point*.

NOTA

Se puede ir viendo el resultado de la textura directamente en el modelo a través de LithUnwrap. Solo tenemos que cargar el modelo en *File/Model/Open*, cargar la textura en *File/Material/Open* y abrir una ventana de *preview* en *Preview/Show Model* (Ver Fig. 10).

En el próximo número...

... además de terminar las texturas del decorado, aprenderemos algunas técnicas para terminar de crear las texturas para obtener texturas propias de diferentes maneras.

Nuestro primer tema musical con Anvil Studio (III)

En el número anterior aprendimos cómo preparar nuestro secuenciador para la grabación MIDI desde un instrumento exterior conectado al ordenador. En esta entrega veremos cómo editar manualmente nuestra composición desde el *Compose*.

SECCIÓN COMPOSE

Como sabemos, es necesario pasar a la sección *Compose* si queremos editar nuestras pistas. Pero antes de entrar, carguemos el tema de ejemplo "FugueGM" del CD. Una vez cargado, vamos a practicar con la primera pista. La seleccionamos y pulsamos en el botón Compose. Observamos cómo la pista se muestra en forma de pentagrama o staff. Podemos cambiar el modo de edición y elegir el modo *Piano Roll* pulsando en la flecha al lado del campo *Staff*. Pero de momento, seguiremos en el modo original. Podemos elegir directamente otra pista utilizando los cursores arriba y abajo y navegar por los compases con los cursores derecho e izquierdo. Para navegar a lo largo del pentagrama con pasos más pequeños, desplazamos la mano que señala la posición actual con el ratón.

EDITANDO EL PENTAGRAMA

Vamos a hacer algunos cambios en la pista del tema que hemos cargado a través del pentagrama.

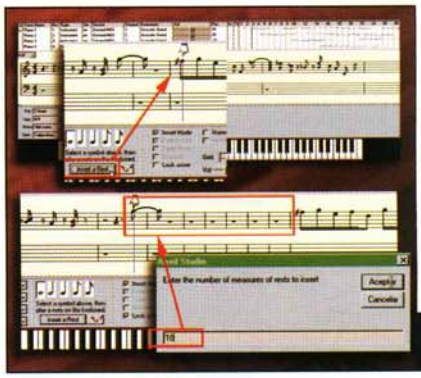
■ Opciones de pentagrama. Situado en la parte inferior del *Compose*, encontramos una serie de opciones que explicamos a continuación. A la izquierda, podemos ver cuatro casillas: *Key*, *Time*, *Show* y *Style* (Ver Fig. 1).

La primera sirve para cambiar la tonalidad de la pista La segunda, nos permite elegir el tiempo. Para ello, nos situamos en el primer compás y elegimos, por ejemplo, 2/4. Si queremos insertar un nuevo tiempo, solo tenemos que situarnos en el compás que lo llevará y elegirlo de la lista. En la casilla *Show* seleccionamos cómo queremos que se muestren las notas. Y la última casilla se utiliza para elegir el estilo que queremos que tenga el pentagrama; es decir, bajos (clave de FA), pentagrama de piano (FA y clave de SOL), doble bajo, doble agudo, etc. En la parte más a la derecha de las opciones tenemos una casilla llamada *Grid*. Aquí, Anvil Studio nos permite ver más divisiones en cada compás. Por ejemplo, si estamos en un 4/4 y seleccionamos *8th notes*, cada compás se dividirá en 8 partes (una para cada corchea).

■ Insertando silencios, notas y acordes. Vamos a continuar haciendo algunos ejercicios. Comenzamos, insertando, por ejemplo, un compás en blanco a partir del primero. Para hacer esto tenemos que tener en cuenta que insertaremos un silencio equivalente a la nota que tengamos seleccionada en la paleta de notas. Así que, si estamos en un 4/4, un compás de silencio correspondería a una "redonda". Nos situamos en el segundo compás, pulsamos en la nota "redonda" y sobre el botón Insert a Rest. Para insertar un número determinado de compases en blanco, pulsamos dos veces sobre el botón de "insertar" y aparecerá una ventana de diálogo pidiendo el número de ellos. Para borrarlos, solo tenemos que seleccionarlos con el ratón y luego pulsar la tecla de "suprimir" del teclado (Ver Fig. 2).

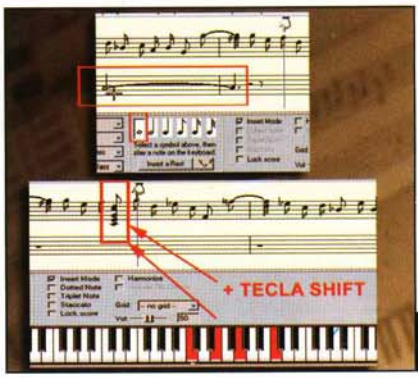
1

Esquema de las opciones de la sección Compose.



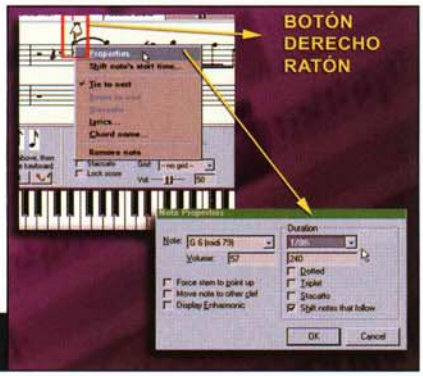
2

Con la opción Insert a Rest es posible insertar silencios y múltiples compases vacíos.



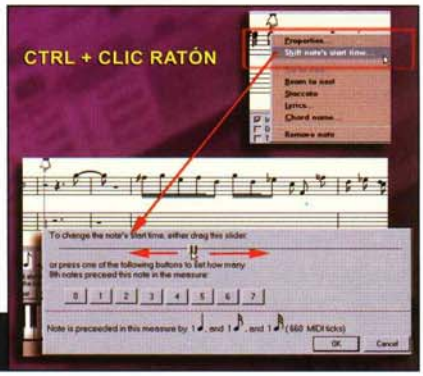
3

Manualmente o mediante el teclado virtual podemos insertar notas y acordes.



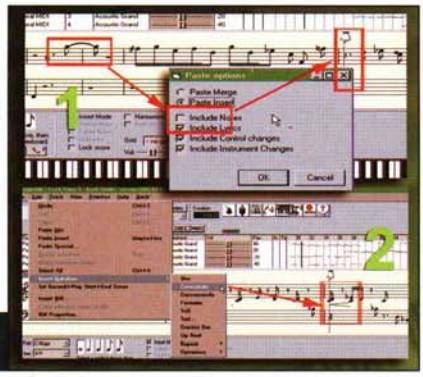
4

Pulsando con el botón derecho del ratón sobre la nota podemos elegir y modificar sus propiedades como volumen o duración.



5

Desplazar las notas hacia ambos lados resulta más fácil manteniendo pulsada la tecla "Ctrl".



6

En la zona 1 se puede observar el resultado de copiar sólo los atributos de la nota. En la zona 2 se muestra el menú de opciones de Insert Notation.

Si en vez de compases en blanco o silencios, queremos insertar una nota, solo tenemos que desplazar con el ratón cualquiera de ellas desde la paleta de notas hacia el pentagrama. El puntero cambiará a una cruz, la cual nos indica en qué posición vamos a situar la nueva nota. Una forma más fácil de insertar notas sería utilizando el teclado virtual que se muestra en pantalla. En primer lugar, seleccionamos dónde vamos a insertarla, después elegimos de la paleta el tipo de nota que queremos insertar y por último, solo tenemos que tocarla en el teclado. También es posible añadir acordes completos. Para ello, debemos dejar pulsada la tecla "Shift" mientras colocamos o tocamos las notas (Ver Fig. 3).

Otra opción interesante que Anvil Studio nos brinda es poder insertar notas desde un micrófono conectado a la tarjeta de sonido. Para realizar esta operación, activamos nuestro micro y cantamos. Seguidamente, pulsamos sobre el icono .

Cambiando propiedades. Vamos a continuar cambiando las propiedades de cualquier nota, por ejemplo, el volumen o la duración. Para abrir la ventana flotante de propiedades, pulsamos con el botón derecho del ratón sobre la nota que queremos modificar y elegimos *Properties*. En esta ventana flotante encontramos las opciones que queremos modificar (Ver Fig. 4).

Desplazando notas. Una vez insertada la nota, podemos desplazarla por el pentagrama hacia arriba o hacia abajo mediante el ratón. Para hacerlo hacia la derecha o izquierda pulsamos con el botón derecho del ratón sobre la nota para entrar en la opción *Shift note's start time*. Encontramos un deslizador (y su equivalente en botones) para asignar el desplazamiento (Ver Fig. 5).

Si queremos oír la nota mientras la desplazamos, no olvidemos dejar activada la opción mediante el icono .

Copiar grupos de notas. Para copiar notas de un lugar a otro, las seleccionamos con el ratón o mediante los cursores más la tecla "Shift" y pulsamos "Ctrl" + "C". A continuación, disponemos de tres tipos diferentes de copiado: por mezcla, por inserto o modo especial (todas estas opciones las encontramos en el menú *Edit*). El modo de mezcla (*Paste Mix*) nos permite crear un acorde; es decir, mezclamos las notas copiadas con las existentes en el lugar de destino. El modo normal (*Paste Insert*) es por inserto, en el que las notas copiadas desplazan a las notas del destino. Y por último, el modo especial (*Paste Special*) nos permite seleccionar qué evento queremos insertar en una ventana flotante de opciones. Si, por ejemplo, deseleccionamos la casilla *Include notes*, a la hora de insertar, solo lo hará el silencio que corresponde al valor de la nota (Ver Fig. 6 / 1).

Insertar anotaciones. También es posible insertar símbolos y anotaciones en el pentagrama como ligados, reguladores, *crescendos*, etc. Por ejemplo, vamos a insertar un *crescendo* en cualquier compás. Nos situamos en él y elegimos la opción *Crescendo* en *Insert Notation* del menú *Edit*. Podemos ver entonces cómo aparece el símbolo elegido (Ver Fig. 6 / 2).

TRUCO

Una manera más rápida de desplazar una nota hacia atrás o hacia delante, consiste en situarnos sobre ella y pulsar la tecla "Ctrl". Sin dejar de pulsar movemos el deslizador con el ratón directamente. Al dejar de pulsar "Ctrl" la nota se desplaza.

En el próximo número...

... aprenderemos cómo editar en el modo *Piano Roll*. Además, también empezaremos a trabajar con las pistas de audio.

Manejo de luces en Blitz3D

Como en el mundo real, toda escena 3D necesita de una iluminación para poder ser apreciada por la cámara. Un uso adecuado de las luces puede cambiar la ambientación del entorno notablemente.

Por ejemplo, no se puede conseguir un ambiente lúgubre si no disponemos de tonos oscuros en la iluminación. Tampoco podemos simular la luz del Sol sin tonos cálidos y brillantes. Por eso, un buen criterio en la elección de las luces hará que nuestra escena 3D gane realismo.

Blitz3D nos proporciona las instrucciones necesarias para lograr la iluminación que necesitamos. Podemos crear diferentes tipos de fuentes luminosas y alterar su color, rango o potencia. Además, las luces son tratadas también como entidades, así que podemos aplicarles alguna de las funciones de éstas como posición, desplazamiento o factores de visibilidad.

En Blitz3D, las luces determinan el color de todos los vértices de un polígono. La influencia que la luz tendrá en el entorno influirá por su rango de acción.

LUZ AMBIENTAL

Por defecto, Blitz3D establece una luz ambiental que ilumina a todos los vértices de los objetos por igual y que afecta a todas las demás luces. Disponemos de una instrucción que nos permite cambiar el tono de esta luz: "AmbientLight".

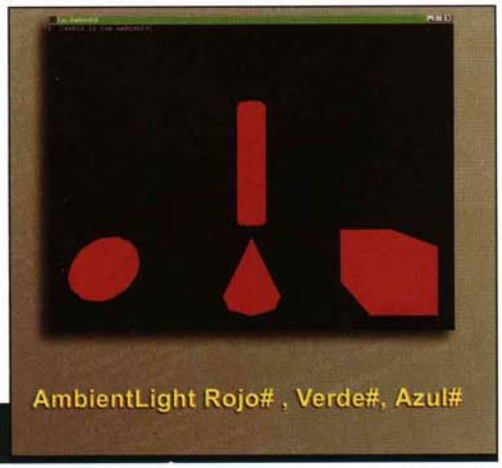
```
AmbientLight Componente de color Rojo#, Verde#, Azul#
```

Por ejemplo, para determinar una total oscuridad escribiremos:

```
AmbientLight 0, 0, 0
```

Si solo usásemos esta luz, se verían todos los objetos de la escena sin ningún tipo de sombra, es decir, planos (Ver "ejemplo1.bb") (Fig. 1).

Para evitarlo, debemos crear al menos algún tipo de luz diferente.

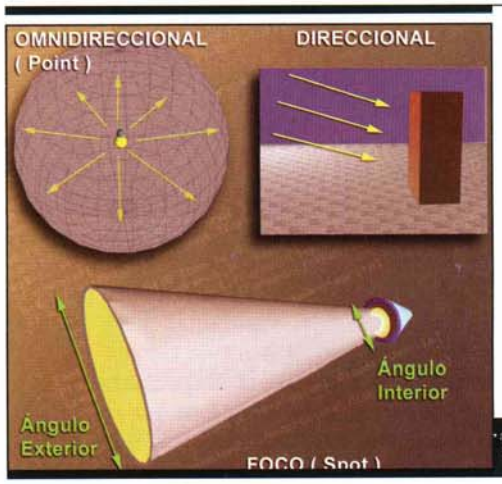


1

Por defecto, Blitz3D nos proporciona una luz ambiental, la cual afecta a todos los polígonos por igual y que podemos cambiar de tono de color.

CREANDO LUCES

En Blitz3D podemos crear tres tipos de fuentes de luz: direccional, punto de luz y foco de luz. Para crear cualquiera de estos tres tipos, disponemos de la instrucción "CreateLight":



2

Para evitar que los objetos de la escena se vean planos necesitamos crear algún tipo de luz. En la figura se muestra el esquema de las diferentes luces que podemos crear en Blitz3D.

```
Luz = CreateLight ( [Tipo de luz] [,entidad madre] )
```

Los tipos de luces se determinan por un número:

- **Tipo de Luz = 1** ➡ Luz direccional
- **Tipo de Luz = 2** ➡ Luz Puntual
- **Tipo de Luz = 3** ➡ Luz Focal

Podemos también especificar una entidad madre para la luz; es decir, si tenemos un cubo y lo asignamos como entidad madre, al mover el cubo la luz se moverá con él:

```
Cubo=CreateCube()
Luz=CreateLight(2,Cubo)
...
MoveEntity Cubo,1,0,0
...
```

Por defecto, si no asignamos ningún número al parámetro "Tipo de luz", se creará una luz direccional (Fig. 2).

LUZ DIRECCIONAL

Este tipo de fuente de luz ilumina un área completa por igual dependiendo del ángulo.

3

En el ejemplo número 2 podemos observar cómo afecta a los polígonos de los objetos 3D una luz direccional en rotación.

Generalmente, se utiliza para simular la luz del sol. A diferencia de la luz ambiental, la luz direccional no ilumina toda la escena sino solo los polígonos de los objetos, los cuales recibirán la luz desde la misma dirección y con la misma intensidad. Modificando el ángulo de la fuente de luz se pueden conseguir variaciones de iluminación, por ejemplo, en un plano (Ver "ejemplo2.bb") (Fig. 3):

```
Luz_direccional = CreateLight (1)
```

Este tipo de luz con un ángulo de inclinación es ideal para iluminar terrenos. El resultado de la inclinación será más visible si se activa el "shading" (sombreado) del terreno (Ver "ejemplo3.bb) (Fig. 4).

4

Si activamos el "shading" del terreno podemos observar cómo varía la incidencia de una luz direccional cuando cambiamos su ángulo.

LUZ OMNI-DIRECCIONAL

La luz omnidireccional o puntual es un punto de luz similar, por ejemplo, a una bombilla. Se basa en crear una fuente de luz en un punto determinado, propagándose en un radio de 360 grados hasta alcanzar el límite establecido. A medida que se extiende, va perdiendo intensidad generando un degradado de luz:

```
Luz_Omni = CreateLight( 2 )
```

(Ver "Ejemplo4.bb") (Fig. 5).

FOCO DE LUZ

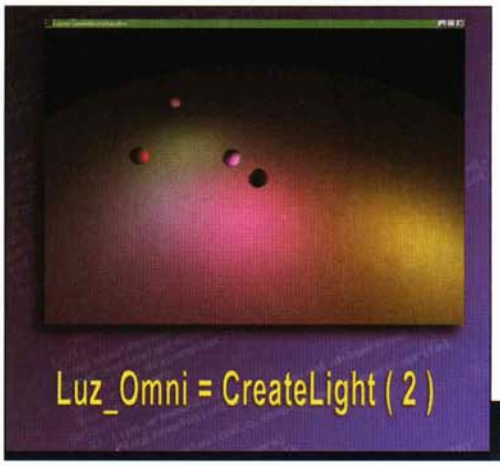
El tercer tipo de iluminación que podemos crear con "CreateLight" simula un proyector; es decir, un cono de luz.

```
Proyector = CreateLight ( 3 )
```

Para modificar la forma cónica se dispone de dos círculos de luz, uno interior y otro exterior, cuyo tamaño puede ser modificado. Ahora bien, este tamaño viene establecido por la apertura de ambos conos, la cual es determinada por un ángulo. Para proporcionar los valores a ambos ángulos, disponemos de la instrucción "LightConeAngles":

```
LightConeAngles Luz,
Angulo interior#,
Angulo exterior#
```

Los valores por defecto se sitúan en un ángulo mínimo de 0 y uno máximo de 90 grados. El valor máximo posible para el anillo interior va desde 0 hasta el ángulo del anillo exterior.



5

Para mostrar adecuadamente el aspecto completo de las luces omnidireccionales necesitamos trabajar en una escena de gran tamaño.

(Ver "ejemplo5.bb", "ejemplo6.bb" y "ejemplo7.bb") (Figs. 6, 7 y 8).

En ocasiones, necesitaremos que un foco apunte siempre hacia un objeto de la escena. Por ejemplo para iluminarlo mientras éste se mueve. Para lograrlo, tendremos que utilizar la instrucción "PointEntity":

```
PointEntity Luz, Objeto3D
```

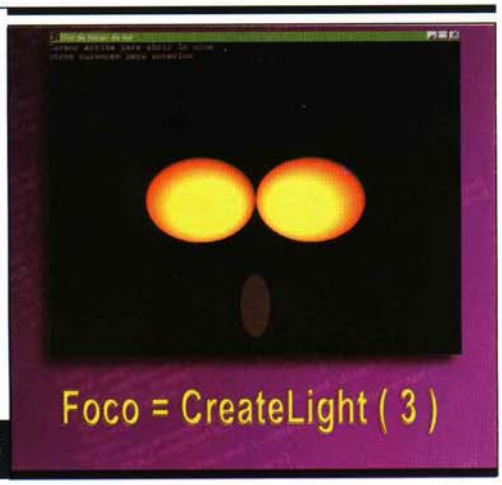
OPERACIONES CON LAS LUCES

Los tres tipos de luces pueden posicionarse o desplazarse por la escena 3D por medio de las instrucciones "PointEntity" y "MoveEntity". Además, se les puede asignar cualquier color con la instrucción "LightColor":

```
LightColor Luz, Componente de
color Rojo#, Verde#, Azul#
```

NOTA

Es preciso recordar que Blitz3D permite crear simultáneamente todas las luces que la tarjeta de vídeo soporte; generalmente, un máximo de 7 (con la luz ambiental por defecto incluida).



6

Se pueden realizar algunos efectos artísticos usando focos.

```
LightRange Luz
Rango#
```

No hay establecido un valor exacto para el rango de cada luz. Éste es aproximado y pasa por aplicar el ensayo - error para obtener los resultados más adecuados.

Una opción muy interesante que nos ofrece Blitz3D es la de crear una falsa iluminación sobre objetos 3D (*meshes*). Este sistema resulta interesante para "pintar" nuestros decorados. Para realizar esta operación disponemos de la instrucción "LightMesh":

```
LightMesh Mesh, Componente de
color Rojo#, Verde#, Azul#
[,Rango][,PosX_Luz][,PosY][,PoZ]
```

7

En el ejemplo 6 se aplica el desplazamiento de un foco para simular la representación de un eclipse de sol.

Asignando un color blanco ("LightColor Luz, 255, 255, 255") establecemos el brillo máximo para esa luz. Por el contrario, con un color negro ("LightColor Luz, 0, 0, 0") no habrá efecto de brillo alguno. También, se puede obtener luz negativa aplicando a cada componente de color un valor negativo máximo ("LightColor Luz, -255, -255, -255"). Esto anula cualquier tipo de brillo. Es ideal para obtener efectos de sombras. En el ejemplo 8 se puede observar cómo proyectando una luz negativa desde la esfera, dicha luz origina la sombra de ésta sobre el suelo (Fig. 9).

Desgraciadamente, este tipo de técnicas no resulta eficaz en la práctica si queremos proyectar sombras irregulares o de varios objetos al mismo tiempo, ya que necesitaríamos una luz para cada uno de ellos y, lógicamente, existe un límite impuesto por nuestra tarjeta gráfica.

Como comentábamos en la introducción, la influencia de las luces viene determinada por su rango de acción. Todo lo que queda fuera de ese rango no será afectado por la iluminación. Para modificar este aspecto, Blitz3D nos proporciona la instrucción "LightRange":

UTILIZACIÓN DE LUCES

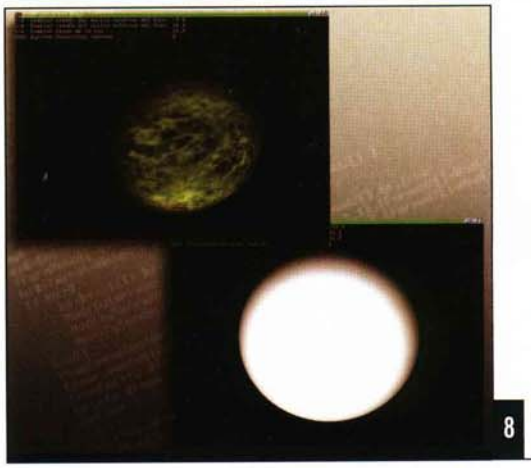
El uso de luces se torna variadísimo en multitud de ocasiones. Para los videojuegos, su utilidad es evidente y entre una lista interminable de posibilidades se me ocurren algunas interesantes y fáciles de implementar. Para ello, vamos a recorrer cada una de las diferentes instrucciones que disponemos para el desarrollo de iluminación.

NOCHE Y DÍA

Este efecto resulta ideal para simular realismo en entornos abiertos. Es sumamente fácil de programar y el resultado "salta a la vista". El truco radica en utilizar las propiedades que nos ofrecen la instrucción "AmbientLight" y las luces direccionales. En primer lugar, es necesario crear una luz por defecto (tipo 1, direccional) que hará el papel del sol. Seguidamente, debemos utilizar una variable del tipo *float* que utilizaremos para aplicarle rotación:

```
;Definiciones
Sol = CreateLight()
Rotación_Luz# =0.0
...
```

Debemos jugar, además, con la luz ambiental, variando su brillo para aumentar el efecto de la rotación del sol. Así que tenemos que utilizar otra variable *float* para disminuir y aumentar por igual las componentes de color:



8

Podemos observar el funcionamiento de todos los parámetros de una luz focal en el ejemplo número 7 contenido en el CD.

9

Por medio de luces negativas es posible obtener sombras dinámicas en tiempo real.

```
;Definiciones
Luz# = 200.0  ; Día
AmbientLight Luz#, Luz#, Luz#
```

Para lograr que la luz se modifique desde un valor a otro y viceversa, utilizaremos otra variable que servirá de incremento de la cantidad de luz (variable "Luz"). Por último, crearemos una función que hará todo el proceso y que tenemos que debemos llamar desde el bucle principal del juego:

```
Function Dia_Noche()
  If Luz<10 Variación_Luz=
  (Variacion_Luz)*-1
  If Luz>254 Variación_Luz=
  (Variacion_Luz)*-1
  Luz=Luz- Variación_Luz
  AmbientLight Luz, Luz, Luz
  Rotacion_Luz = Rotación_Luz + .1
  RotateEntity Sol, Rotación_Luz, 0,0
End Function
```

TRUCO

Se pueden utilizar las ventajas de la función coseno para lograr un modo más elegante de fluctuación entre claridad y oscuridad, evitando el uso de condicionales:
Iluminación_Máxima=50 :
Velocidad_Transición#=0.01
Luz=Iluminación_Máxima + Cos (MilliSecs())*
Velocidad_Transición

Con las dos sentencias condicionales, simplemente, cambiamos el signo de "Variación_Luz" para conseguir sumar o restar el incremento del valor de "Luz". Con ello, conseguimos fluctuar de un valor a otro.

EXPLOSIONES Y DISPAROS

El segundo tipo de iluminación (omnidireccional) nos puede ayudar a enfatizar, aún más, las explosiones de nuestros juegos. Simplemente, consiste en ir creando y destruyendo luces en el momento del impacto. Realmente, si vamos a tener un uso masivo de explosiones en el juego puede ocurrir que, en ocasiones, se rebase la barrera del máximo de luces simultáneas permitido por la tarjeta de vídeo. Llegado el caso, veremos cómo nuestra escena se oscurece por momentos, ya que perdemos la luz ambiental. Para evitar este problema, es fundamental destruir la luz una vez visualizada. Aunque este sistema tiene una efectividad del 70%, siempre se puede evitar utilizar luces reales para este cometido y simularlas por medio del coloreado o pintado de los triángulos involucrados en la explosión. Esta técnica de mapeado de luces en tiempo real denominada "*vertex lighting*" resulta algo más complicada de implementar, pero es una de las alternativas más usadas por los juegos de hoy día.

De igual manera, es posible aplicar este efecto a los disparos. Básicamente, creando una luz en el momento de disparar, por ejemplo, para simular la combustión del arma. Por razones que ya hemos comentado, no es conveniente crear una luz y que se desplace en la trayectoria del disparo. Fácilmente, sobrepasaríamos el límite de luces.

LINTERNAS

Este tipo de efecto es fácil realizarlo utilizando el tercer tipo de luz (foco). Consiste en situar en el objeto emisor (linterna) un foco de luz con el ajuste del rango y el ángulo de apertura adecuados. Podíamos ver el uso de esta técnica en el tutorial que dedicamos a los niveles BSP del número 11. En ella, situamos la linterna al nivel de la cámara:

```
Camara = CreateCamera()
Linterna=CreateLight(3, camara)
LightConeAngles Linterna, 0, 60
...
```

SOMBRAS DINÁMICAS UTILIZANDO LUCES

Como hemos comentado, existe una técnica para generar sombras en tiempo real de un objeto 3D mediante el uso de luces negativas. Básicamente, la teoría consiste en asignar al objeto un foco de luz negativa (sombra), el cual se orientará hacia la fuente de luz principal (Ver "ejemplo8.bb") (Fig. 9).

En el próximo número...

... aprenderemos cómo utilizar el teclado, ratón y dispositivos para juegos en nuestros programas.

10

Ejemplo de iluminación por medio de un foco simulando una linterna.

Creación y organización de tablas de récords

Una tabla de récords es algo casi fundamental en un juego arcade, en el que el jugador acumula puntos.

Básicamente, consiste en mostrar una lista, más o menos extensa, de nombres con los correspondientes puntos conseguidos. Para que un jugador pueda inscribir su nombre en la lista, es necesario que acumule una cierta cantidad de puntos superando una cifra mínima preestablecida. La lista puede ser todo lo grande que se quiera. Normalmente, se visualizan grupos de diez nombres en pantallas consecutivas, pero a menudo se suele usar un scroll vertical para mostrar toda la lista. En nuestro tutorial vamos a implementar una de estas tablas. La lista contendrá sólo diez jugadores y la puntuación mínima inicial para poder entrar en la tabla es de 3000. La idea es mostrar a la vez en pantalla únicamente las diez mejores puntuaciones ordenadas de mayor a menor. Cuando un jugador supere el mínimo se insertará en la tabla en el lugar correspondiente. Debemos entonces afrontar un par de problemas. Por un lado tenemos que ordenar la tabla de mayor a menor y por otro insertar un elemento nuevo si llega el caso. Para resolver el primero de los problemas se puede utilizar cualquier método de ordenación estándar que veremos a continuación.

MÉTODOS DE ORDENACIÓN

Basaremos la ordenación de la tabla en los valores de puntuación. Existen varios métodos para ordenar una tabla. Entre los más conocidos se encuentran: burbuja, selección, shell, inserción y *quicksort*. Nosotros sólo veremos dos: el método *shell* y el de burbuja.

El resultado es exactamente igual utilizando uno que otro, sin embargo, existe una relativa diferencia de velocidad entre ambos.

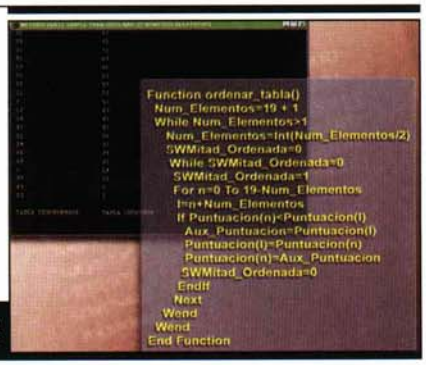
El método shell resulta eficaz para ordenar una tabla con muchos elementos, pero es más complicado de utilizar.

MÉTODO SHELL

Este sistema de ordenación se utiliza para tablas con un gran número de elementos. Por supuesto, no es el ideal para nuestro caso. Además, es mucho más complicado de entender que el de burbuja. Sin embargo, lo hemos incluido por motivos didácticos. Este método, básica-

Código 1. Método shell

```
Function ordenar_tabla()
 Num_Elementos=10 + 1
 While Num_Elementos>1
  Num_Elementos=Int(Num_Elementos/2)  ; Dividimos la tabla en dos partes
   SwMitad_Ordenada=0
   While SWMitad_Ordenada=0  ; Procedemos a recorrer y ordenar cada mitad
   SWMitad_Ordenada=1 ; Activamos en Switch
   For n=0 To 9-Num_Elementos ; Recorremos la primera mitad para ordenarla
    I=n+Num_Elementos ; Situamos el puntero de comparación
    If Puntuacion(n)<Puntuacion(I)
     ; Intercambiamos los elementos de la tabla para ordenarlos
     Aux_Puntuacion=Puntuacion(I): Aux_Jugador$=Jugador$(I)
     Puntuacion(I)=Puntuacion(n) : Jugador$(I)=Jugador$(n)
     Puntuacion(n)=Aux_Puntuacion: Jugador$(n)=Aux_Jugador$
     SWMitad_Ordenada=0 ; Hasta que esta mitad no este ordenada no pasamos a la siguiente
    EndIf
   Next
  Wend
 Wend
End Function
```



2

En el ejemplo "Shell.bb" se puede observar el funcionamiento del método shell con una tabla de 20 números aleatorios.

mente, lo que hace es dividir la tabla en dos. Una vez que ha ordenado una parte, vuelve a dividir la que queda sin ordenar. Y así sucesivamente hasta que no pueda dividir más. Para crear las partes nos basamos en una variable que controla el número de elementos de cada una de ellas llamada *intervalo* (Ver "shell.bb") (Código 1).

La variable "Num_Elementos" es la que utilizamos como intervalo y "SWMitad_Ordenada" es un Switch que nos avisa si la parte está totalmente ordenada.

MÉTODO DE BURBUJA

Este método es mucho más lento para ordenar gran cantidad de elementos, pero resulta ideal para nuestra tabla de récords, ya que sólo tenemos que ordenar diez elementos. Además, es muy sencilla de comprender, así que será el que utilicemos para nuestro juego. Consiste en comparar cada elemento con todos los demás. Si están desordenados, se intercambian en cada comparación para ir situando el valor mayor sobre el menor. De esta forma, se puede observar cómo los valores mayores van subiendo hacia la parte superior de la tabla; de ahí el nombre de "método de burbuja" (Ver Código 2).

Para ir comparando un elemento con todos los demás necesitamos dos bucles anidados. En el bucle interior vamos comparando un elemento (bucle exterior) con todos los demás (bucle interior) e intercambiándolos si uno es mayor que el otro. Para ordenar de menor a mayor sólo es necesario cambiar el operando de condición "<" por ">":

Código 2. Método de la burbuja

```
Function ordenar_tabla()
 For n= 0 To 9
  For m= 0 To 9
   If Puntuacion(m) < Puntuacion(n)
    ; Intercambiamos valores
    Aux_Puntuacion=Puntuacion(m): Aux_Jugador$=Jugador$(m)
    Puntuacion(m)=Puntuacion(n) : Jugador$(m)=Jugador$(n)
    Puntuacion(n)=Aux_Puntuacion: Jugador$(n)=Aux_Jugador$
   EndIf
  Next
 Next
End Function
```

Código 3. Insertamos un nuevo récord

```
Function Insertar_en_Tabla(j$,p)
 Aux_Puntuacion=p : Aux_Jugador$=j$
  For n=9 To 0 Step -1
   If Aux_Puntuacion>Puntuacion(n)
    Puntuacion(n)=Aux_Puntuacion: Jugador$(n)=Aux_Jugador$
    Return
   EndIf
  Next
End Function
```

3

Para insertar una nueva puntuación en la tabla se puede utilizar un método de inserción simple, donde el nuevo elemento desplaza al último de la tabla.

```
If Puntuacion(m) < Puntuacion(n)
➡ De Mayor a Menor
If Puntuacion(m) > Puntuacion(n)
➡ De Menor a Mayor
```

INSERTANDO UN NUEVO RÉCORD EN LA TABLA

Ya tenemos solucionado el tema de la ordenación de los récords de mayor a menor. Ahora solo nos queda utilizar un método para añadir un nuevo récord en la tabla. Para ello utilizaremos el método de inserción (Ver. "Inserción.bb") (Ver Fig. 3) (Ver Código 3).

Una vez insertado el nuevo récord, volvemos a ordenar la tabla por el método de burbuja.

Fundamentalmente, lo que hacemos es añadir un elemento (j$ y p) al final de la tabla desplazando el valor más pequeño de la misma. Así que tenemos que buscar la nueva posición del elemento empezando por el final. Si el número insertado ("Aux_Puntuación") es mayor que el de la tabla ("Puntuación(n)") colocamos el nuevo número en su lugar.

Si es encontrado, colocamos ahí el nuevo récord y salimos del bucle.

En el próximo número...

... aprenderemos un sistema para grabar y reproducir los movimientos de la cámara. Ideal para realizar presentaciones de nuestro juego con el motor del Blitz3D.

Juegos de rol o JDR (RPG)

En esta entrega vamos a centrarnos en otro de los grandes géneros del mundo de los videojuegos para ordenador: los juegos de Rol o JDR (RPG).

En este tipo de producciones, el jugador toma la identidad de un personaje o varios para participar en una aventura mágica. Hay infinidad de juegos de este género para PC, así que veremos solo los más representativos desde los comienzos.

HISTORIA Y EVOLUCIÓN

Prácticamente todos los juegos de rol para ordenador siguen las reglas básicas *Dungeons & Dragons* (Dragones y Mazmorras) publicadas por Dave Amerson y Gary Gygax en 1974. Fueron creadas a partir de los *Wargames* clásicos que se jugaban con tablero, fichas y figuras de dragones, enanos y otras criaturas. La diferencia para estas nuevas reglas radicaba en que el jugador solo controlaba a un personaje y no a un ejército. En la década de los 80, los juegos de rol pasaron de los libros de lucha y tableros al ordenador para acercar al público en general a este género tan carismático. Los primeros juegos pasados a ordenador se basaban en texto sin ningún tipo de gráficos, algo parecido a como se haría con un libro. Sin embargo, aun avanzada la tecnología, el espíritu de los juegos de rol se ha mantenido intacto. Antes de que los ordenadores hubieran dominado el mercado lúdico, eran las consolas las que ofrecían todo tipo de géneros y es obligatorio hablar de los primeros RPG para consolas que siguieron las reglas *D* & D como *Quest For The Rings* publicado para Odyseey o *Cloudy Mountains* para Intellivision a comienzos de los 70. Al principio, estos juegos mezclaban la acción con ciertos toques de rol. Fueron Nintendo y Sega los que dominaron el mundo RPG con el lanzamiento de *Zelda* y la serie *Final Fantasy* con unos gráficos e historia increíbles para la época. Generalmente, la perspectiva usada era cenital pero había títulos que simulaban las 3D pantalla a pantalla como *Shining in the Darkness*. Las producciones para consolas siguen hoy en día con gran éxito, y quizás sea *Final Fantasy* la más famosa y aclamada. Sin embargo, el dominio del PC hizo que los RPG adquirieran otra dimensión en gráficos y jugabilidad. Aunque una de las producciones para ordenador que más caló en el público fue la serie *Ultima* creada por Richard Garriot en 1980. Garriot desarrolló 9 títulos, todos siguiendo los avances tecnológicos del momento. Desde la representación en blanco y negro de la primera entrega, pasando por la vista cenital clásica, hasta la representación 3D a partir de *Ultima Underworld*. Pero para clasificar a los RPG más sobresalientes de la historia para PC debemos guiarnos por los que siguen las reglas "Dungeons & Dragons" y los más avanzados "Advanced D&D".

DUNGEONS & DRAGONS Y ADVANCED DUNGEONS &DRANGONS

Las reglas *D &D* se basan en que el jugador encarna a un personaje y éste va aumentando de experiencia y poder durante el juego. Básicamente *AD & D* es igual que su antecesor pero incorporaba más reglas y subreglas triunfando a mediados de los 80. Durante más de una década se

1

Los RPG mas famosos para consola: Zelda y Final Fantasy.

2

Diablo I y II. Clásicos indiscutibles del género.

3

(Arriba) Dungeon Siege, exquisito entorno 3D real. (Abajo) Nox intentó competir con Diablo sin conseguirlo.

4

Juegos de Rol modernos. (Arriba) Neverwinter Nights de Bioware con editor de juegos incluido. (Abajo) Morrowind: The Elder scrolls III, la última entrega de la serie The Elder scrolls.

LA BIOGRAFÍA...

RICHARD GARRIOTT

Creador de la serie **Ultima**

Después de vender 30.000 copias de su primer juego de rol *Akalabeth* para Apple II en 1980 crea el primer título *Ultima* y funda la compañía Origin System con su hermano Robert. Amante de los juegos de rol en todas sus vertientes, lucha por conseguir introducir a los jugadores en mundos de fantasía más allá de los prejuicios mundanos. Es uno de los desarrolladores más veteranos y respetados de la industria gracias a su trabajo con la serie *Ultima*. Llegó a convencer con las 3D de *Ultima IX Ascension*. Actualmente ya no está en Origin y está totalmente dedicado a *Ultima: Online 2* totalmente en 3D.

Richard Garriott.

ha intentado recrear la experiencia de los juegos *D&D* de lápiz y papel en ordenador. Fue la empresa *SSI* (Strategic Simulations Incorporated) la que primero logró realizar algo positivo y lo hizo *con Pool of Radiance* (1988) y más de una docena de títulos más como: *Heroes of the Lance* (1988), *Secret of the Silver Blades* (1990), el fantástico *Eye of the Beholder I* (1991), primer juego basado en *AD&D* en primera persona con un motor gráfico en pseudo tiempo real.

Siguieron otros títulos basados en la idea de *Eye of the Beholder* como: *Dungeon Hack* (1993) o *Menzoberrazan* (1994). Al margen de SSI, otras desarrolladoras destacaban por la calidad de sus juegos considerados clásicos del género como *SepterraCore: Legacy of the Creator* (Valkyrie Studios, 1999) o una de las producciones que consagró el género de la franquicia *AD&D* y que implantó una forma de hacer juegos de Rol fue *Baldur´s Gate* en 1998, publicado por InterPlay y desarrollado por BioWare. Anterior a estos títulos, una desarrolladora muy especial con un talento increíble y un equilibrado en los juegos inmejorable llamada Blizzard Entertaiment publica *Diablo*. Ambientado en el mundo medieval, sigue las reglas básicas de *D & D*. Era la primera incursión de Blizzard en Windows y poseía unos gráficos bien acabados con solo 256 colores. Su concepto de jugabilidad es simple y llega a multitud de usuarios de todo el mundo. Continuó la saga con *Diablo II* (2000). No tan original pero con una jugabilidad envidiable y con gráficos grandes y detallados. Los juegos de rol para PC cobran fuerza en el año 2000 con títulos de gran belleza gráfica. Clásicos como *Pool of Radiance* continúan su andadura por el universo de los Reinos Olvidados en 2001 con *Ruins of Myth Drannor*. Otra desarrolladora de renombre en los juegos de estrategia entra en el mundo del RPG con un título que pretendía desbancar el liderazgo de *Diablo*, nos referimos a Westwood Studios con *Nox* (2001). Fácil de jugar y con un motor gráfico más que aceptable casi alcanza su cometido. Ese mismo año Microsoft decide distribuir un título de *Gas Powered Games* que rompe moldes al combinar lo mejor de otros títulos de renombre y la potencia gráfica de las nuevas tarjetas de vídeo. Nos referimos a *Dungeon Siege*. Siguiendo las reglas *AD & D*, este título posee grandes dosis de acción, aventura y rol con gráficos fantásticos y completamente en 3D, y lo que es más importante; jugabilidad. De nuevo Bioware deja el mundo de las dos dimensiones con *Throne of Baal* y entra de lleno en las 3D con *Neverwinter Nights*, uno de los títulos más esperados basado en las reglas *D&D*. En este título poseemos total libertad de movimientos y podemos controlar varios personajes con habilidades diferentes. Además, viene incorporado de un editor de niveles y personajes fantástico. No podemos olvidar *Morrowind: The Elder scrolls III* (Bethesda Softworks/ Ubi Soft, 2002), la última joya del rol actual en primera y tercera persona.

RPG EN INTERNET

No queremos terminar sin hablar de las posibilidades que internet ofrece a los mundos de aventuras creados por los RPG. El género aprovecha las posibilidades de la red para perpetuarse dentro de mundos infinitos online. Es *Ultima Online* una de las que pervive actualmente con fuerza en todo el mundo a través de internet con miles de jugadores encarnando a personajes humanos en mundos de fantasía. Pero no fue *Ultima* el único título que llevó el rol a todo el mundo a través de la red. Encontramos también un título que ofrece al jugador una mayor libertad de movimientos, mejores gráficos y la posibilidad de encarnar a otros personajes no humanos: *Everquest* y *Everquest II*.

En el próximo número...

... hablaremos del género de aventuras gráficas.

Cuestionario Videojuegos 14

Preguntas

1. Modifica la luz ambiental de una escena y crea tres luces de diferentes tipos en Blitz3D.
2. ¿Cómo podemos modificar la intensidad y amplitud de un foco de luz en Blitz3D?
3. ¿Qué es un emisor de partículas?
4. Define una estructura básica para crear un sistema de partículas.
5. ¿Cómo obtenemos una plantilla o template de un modelo para pintar la textura con Paint Shop Pro?
6. ¿Cómo podemos obtener una textura desde Bryce?
7. ¿Cómo podemos insertar cinco compases en blanco dentro del pentagrama de una pista en Anvil Studio?
8. ¿Cómo podemos desplazar una nota del pentagrama hacia atrás o hacia delante en Anvil Studio?
9. Escribe la rutina para realizar una ordenación de veinte números por el método de burbuja simple.
10. Escribe una rutina para insertar un elemento en una tabla de veinte números.

Respuestas al cuestionario 13

▷ **1.** Utilizando la instrucción "CameraViewport":
CameraViewport PosX, PosY, Ancho, Alto

▷ **2.** Convertimos el objeto en "pickeable" con "EntityPickMode" y luego utilizando "CameraPick".

▷ **3.** Utilizando "EntityCollided", "CountCollisions" y "CollisionEntity":

```
If EntityCollided (bionave,ENTIDAD_ENEMIGO)
   For n=1 To CountCollisions(bionave)
      ent=CollisionEntity(bionave,n)
   Next
   For volador.tipo_voladores= Each tipo_voladores
      If volador\entidad_volador=ent  text 0,0,"DETECTADO": Return
   Next
EndIf
```

▷ **4.** Utilizando la instrucción "AlignToVector" y asignándole al sprite el modo 4 de visualización con "SpriteViewMode".

▷ **5.** Mediante el escalado y desplazamiento de los vértices de los polígonos.

▷ **6.** Para poder pintar el modelo en Deep Paint 3D, éste debe tener un mapeado UV y un material asignado.

▷ **7.** Debemos preparar las conexiones MIDI. Luego, asegurar que el programa recibe los eventos MIDI a través de las opciones de puerto MIDI de "Synthesizers".

▷ **8.** Seleccionando la opción "quantize Entire Track" del menú "Track".

▷ **9.**

```
Dim Letra$   (Num_letras)
Restore Alfabeto
For n=0 To Num_letras
   Read Letra$(n)
Next
.Alfabeto
Data "A","B","C","D","E","F","G","H","I","J","K","L","M"
Data "N","O","P","Q","R","S","T","U","V","W","X","Y","Z"
Data "1","2","3","4","5","6","7","8","9","0"
```

10. Utilizando una imagen para cada letra y sustituirla una vez hallada dentro de la frase:

```
For n = 1 To Len ( Frase$ )
   For m=0 To Num_letras
      If Mid(Frase$,n,1)=Letra$(m)
         GLetra=Grafico(m)
         DrawImage GLetra,x,y
      EndIf
   Next
Next
```

Contenido CD-ROM 14

▶ AUDIO

■ Audio Phonics Guitar Tuner 1.02
Si aún compones usando una guitarra, gracias a este programa podrás afinarla con rapidez y exactitud.

■ Audio Phonics Instrument Tuner 1.02
Otro afinador de instrumentos, complementario al primero, rápido y preciso.

■ Da Metronome 1.1
Completo metrónomo por ordenador para ayudarte a llevar correctamente el ritmo en tus composiciones.

■ Locator MIDI Sequencer 1.10
Secuenciador MIDI de 256 canales. Tiene un teclado virtual, un mixer y un editor WAV con soporte de efectos Direct-X.

■ Cool MP3 Splitter
Divide MP3 por tiempo o tamaño y une los fragmentos.

■ Tactile12000
Aplicación basada en QuickTime que simula dos bandejas de DJ en 3D para mezcla de audio.

▶ DISEÑO 2D

■ Explosion Graphics 3.8.2
Una forma divertida de ver tus fotografías y de escuchar tus archivos de música.

■ Autoimager
Potente y completísimo procesador de imágenes en masa. Ideal para trabajar rápido con muchas imágenes.

■ Quick Screen Capture 1.2
Práctico y funcional capturador repleto de interesantes funciones.

■ EyeBatch 2.0
Renombra, retoca y procesa imágenes en grupo gracias a esta aplicación.

■ PhotoCleaner 1.3
Arregla los problemas comunes que puedas tener en tus imágenes con un solo clic.

■ PhotoPhilia 1.7
Organiza tus imágenes y aplícales sencillos efectos usando este programa.

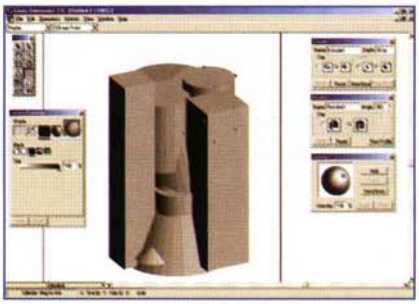
▶ DISEÑO 3D

■ 3D AIM Animation 3.0
Crea animaciones en tres dimensiones usando múltiples modelos.

■ 3D Art Composer 1.1
Construye imágenes en 3D a partir de distintos modelos y fondos.

■ Crossroads
Completo programa para convertir distintos archivos de 3D fácilmente.

■ Adobe Dimensions 3.0
Herramienta poderosa con la que podrás realizar la tarea de renderización de objetos 3D.



■ ModelPress 2.0.0.37
Publica en formato web archivos en tres dimensiones.

■ Terragen 0.8.11
Excelente programa para diseñar paisajes en 3D y renderizar escenas muy realistas.

▶ PROGRAMACIÓN

■ QSetup Installation Suite 3.0
Crea instalaciones sin necesidad de realizar scripts.

■ ImpulseStudio 3.05
Completa suite de componentes Active X para programar en Visual Basic.

■ VTune Performance Analyzer 6.1
Excelente utilidad para analizar datos de diversos modos.

▶ JUEGOS

■ Diablo
Uno de los juegos de rol más populares. Ambientado en la Edad Media, ha llegado a jugadores de todo el mundo.

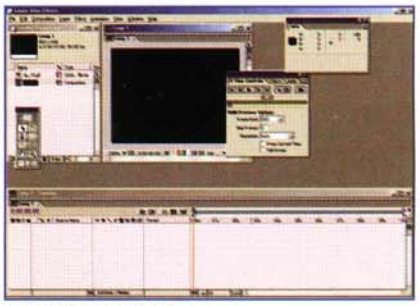
■ Final Fantasy
Uno de los RPG más famoso para consola, en esta versión para PC.

■ Dungeon Siege
Demo del popular juego con exquisito entorno 3D real. Te entusiasmará.

■ Zone of Fighters
Como todas las semanas, nuestro juego.

▶ VÍDEO

■ Adobe AfterEffects 5.5
La herramienta esencial para realizar fantásticos efectos visuales.



■ NVDVD 2.0
Excelente programa de vídeo para que tengas una auténtica experiencia multimedia.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.

Dando vida al terreno de juego

Después de trabajar con el sistema de partículas del juego, pasamos a desarrollar las funciones necesarias para situar todos los elementos del decorado sobre el terreno de combate. Explicaremos el funcionamiento de los dos modos de zona de combate: el aleatorio y el editado.

PETICIÓN DESDE EL MENÚ PRINCIPAL

La elección del tipo de terreno del juego viene determinada por la elección del jugador en el menú principal. Nos situamos en la función "Panel_opciones()" del módulo "Menu.bb". En la opción 0 de la estructura "Select...Case", correspondiente a un nuevo juego, preguntamos al usuario qué tipo de juego desea de entre los dos disponibles:

```
Case 0:  ; Opción nuevo juego
  ...
  Repeat
    ...
    Tipo_juego=Input$(" ")
    If Tipo_juego=0 Then Return
  Until Tipo_juego=1 Or Tipo_juego=2
  ...
```

Con la información almacenada en la variable global "Tipo_juego" ya sabemos qué procedimientos debemos activar para crear el decorado del juego. Cada uno de estos procedimientos lo encontramos en el módulo "funcpantaudio.bb" en la función "Crear_Decorado()".

La función "Crear_Decorado()" es la encargada de llamar a cada uno de los procedimientos necesarios para situar todos los elementos del juego sobre el terreno (construcciones, árboles, rocas, plantas y animales). Se basa en dos estructuras "Select..Case" anidadas, las cuales dirigen la función a los diferentes procesos. La estructura principal selecciona el tipo de juego a través de la variable global "Tipo_juego". Al comienzo de la función "sembramos una semilla" para la generación de números aleatorios diferentes cada vez que se llame a la función:

```
Function Crear_Decorado()
  SeedRnd MilliSecs()
  Select Tipo_juego
    Case 1: ; Zona de juego
    generada por ordenador
...
```

NOTA

De cómo el programa carga y prepara el terreno según el tipo de juego elegido se estudió en el número 9 de esta sección "Visualizando el terreno de juego".

SITUANDO EL DECORADO ALEATORIAMENTE

Si "Tipo_juego" vale 1, el jugador ha elegido generar el decorado aleatoriamente. En el diseño del juego se ha contemplado también distinguir entre los dos tipos de terrenos para generar zonas de combate diferentes. Así que entramos en otra estructura "Select..Case" con las dos opciones de terreno disponibles según el valor contenido en la variable global "Tipo_terreno" (Ver Código 1).

Código 1. Estructura "Select...Case" con las dos opciones de terreno

```
Select Tipo_juego
  Case 1: ; Zona de juego generada por ordenador
    Select Tipo_terreno
      Case 1:
       colocar_almacen1(mesh_almacen1,12000,3000,14000,7500,2*Densidad_decorado,600)
       colocar_almacen2(mesh_almacen2,10000,5000,12000,7500,2*Densidad_decorado,600)
...
  Case 2:
       colocar_arbol1(mesh_arbol1,12000,3000,14000,7500,2*Densidad_decorado,200)
       colocar_arbol2(mesh_arbol2,10000,5000,12000,7500,2*Densidad_decorado,200)
...
End Select
```

Para ejecutar el proceso de colocación es preciso seguir unas pautas si queremos ubicar cada elemento correctamente en su sitio. Por ello, vamos a utilizar una función para cada uno de ellos que hará el trabajo según unos valores predefinidos (Ver Fig.1).

COLOCANDO EL DECORADO

Vamos a utilizar una estructura de datos para cada tipo de objeto para definir su posición. Así que las creamos en el módulo de definiciones "definiciones.bb". Por ejemplo, para los puentes tenemos:

```
Type tipo_puente
  Field x_decorado#,z_decorado#
  Field entidad_decorado
End Type
```

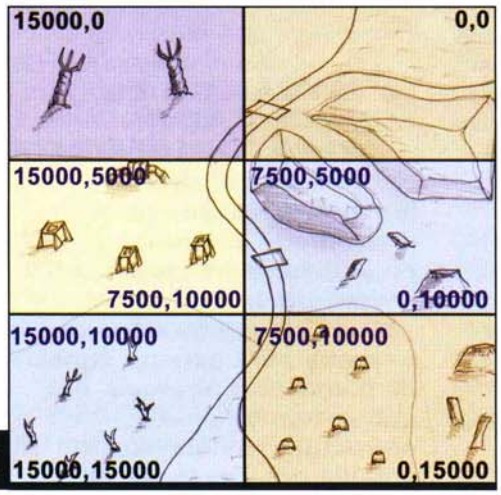
Podíamos haber utilizado matrices (arrays), pero siempre es mejor usar estructuras porque es más rápido, elegante y, a priori, fácil de utilizar para manejar varios datos por objeto. En la estructura se observa que solo contemplamos las coordenadas X y Z, eso es así porque la coordenada Y (altura) la calculamos automáticamente en el momento de colocar el objeto y depende de la altura del terreno, la cual obtenemos con la función "TerrainY". En el módulo "Decorado.bb" instalamos las funciones que sitúan todo el decorado sobre el terreno de juego. Añadimos una función para cada tipo de objeto, la cual agrega un nuevo elemento a la estructura y lo posiciona en el terreno. Vamos a tomar como ejemplo la función para colocar los almacenes de tipo 1 "colocar_almacen1()", ya que la estructura de las demás funciones es exactamente igual. Nuestro propósito para evitar tener que colocar uno a uno cada elemento es disponer de un procedimiento dentro de la función que lo haga automáticamente. Evidentemente, es necesario que le proporcionemos todas las variables necesarias a través de los parámetros de la función. Así que vamos a necesitar los siguientes (Ver Fig.2):

```
Function colocar_almacen1
(entidad,x1_zona#,z1_zona#,
x2_zona#,z2_zona#,cantidad,
dispersion%)
```

Entidad: A través de este parámetro le decimos a la función qué modelo queremos colocar. En realidad, no es necesario, ya que tenemos una función por cada uno de ellos. Aun así, mantenemos flexibilidad en la función y una vez terminada

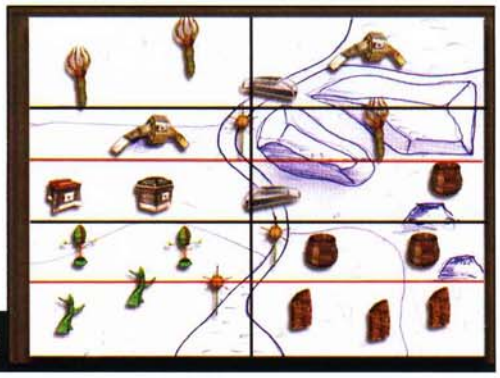
NOTA

Hay que recordar que, sobre el terreno, la coordenada X representa izquierda y derecha y la coordenada Z delante y atrás.



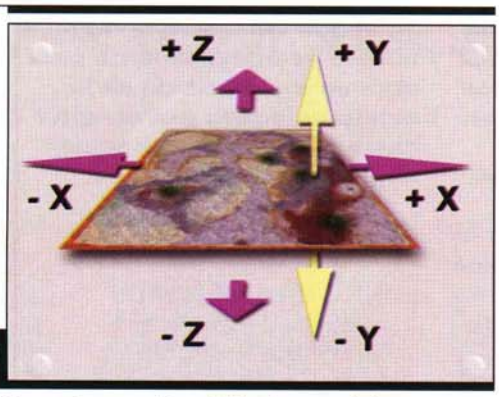
1 *Coordenadas de las seis zonas principales de situación del decorado.*

3 *Esquema del funcionamiento de los parámetros de la función "colocar_decorado()".*



2 *En la imagen vemos subdivisiones creadas para colocar cada tipo de decorado.*

4 *Esquema de los datos leídos y utilizados del fichero de zonas.*



5

Tomando como referencia la cámara, en la imagen se muestra la dirección en las distintas coordenadas.

6

Ejemplo del resultado de aplicar la función para colocar aleatoriamente un modelo.

no necesitaremos volver a entrar para chequearla desde otro módulo.

Para delimitar la zona donde se extenderá la situación del objeto, vamos a definir un rectángulo sobre el terreno. Precisaremos de su posición inicial (definida por la esquina superior izquierda) y su tamaño (esquina inferior derecha). Por lo tanto necesitamos dos coordenadas X y Z para cada punto:

- **X1_zona#:** Coordenada X de la esquina superior izquierda de la zona de colocación.
- **Z1_zona#:** Coordenada Z de la esquina superior izquierda de la zona de colocación.
- **X2_zona#:** Coordenada X de la esquina inferior derecha de la zona de colocación (tamaño).
- **Z2_zona#:** Coordenada Z de la esquina inferior derecha de la zona de colocación.

Luego debemos pasarle a la función la cantidad de objetos que queremos colocar y la separación entre ellos (Ver Fig. 3):

- **Cantidad:** Nº de objetos para situar.
- **Dispersión%:** Separación entre los objetos.

Una vez que tenemos todos los datos necesarios para poder situar los objetos, entramos en la función para realizar la operación. En primer lugar, es necesario crear tantos objetos como hemos indicado en "cantidad". Para ello, entramos en un bucle para ir creando nuevos elementos de la estructura y asignarle a cada uno una copia del *mesh* correspondiente y asignarles la identidad para el sistema de colisiones:

7

El editor de niveles muestra la disposición exacta que tendrá el decorado en el juego.

```
For n = 1 To cantidad
 almacen1.tipo_almacen1=
  New tipo_almacen1
 almacen1\entidad_decorado=
  CopyEntity(entidad)
 EntityType almacen1\
  entidad_decorado,
  ENTIDAD_EDIFICIO
Next
```

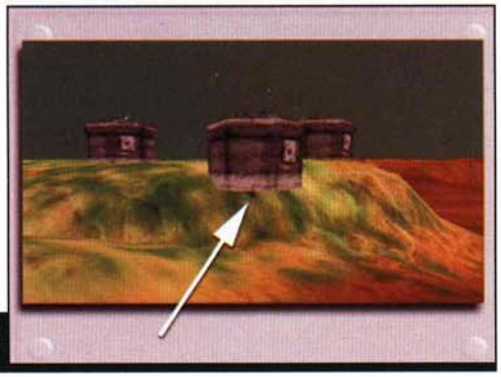
Una vez que hemos creado todos los almacenes pasamos a situarlos en el terreno. Para ello, recorremos cada uno de los elementos de la estructura con un bucle "For .. = Each" (Ver Código 2).

Almacenamos en "m" la entidad que vamos a colocar para reducir código posteriormente. Seguidamente, asignamos unas coordenadas aleatorias de situación dentro de la zona definida por los valores "x1_zona, z1_zona" y "x2_zona, z2_zona". Luego guardamos en las variables "xg","yg" y "zg" las coordenadas 3D del objeto para poderlo posicionar.

La coordenada Y para la altura se calcula automáticamente dependiendo de la del terreno en la coordenada especificada por "xg" y "zg" aprovechando la instrucción

Código 2. Recorremos cada elemento de la estructura

```
For almacen1.tipo_almacen1= Each tipo_almacen1
 m=almacen1\entidad_decorado
 almacen1\x_decorado#=Rnd(x1_zona#,x2_zona#)
 almacen1\z_decorado#=Rnd(z1_zona#,z2_zona#)
 xg#=almacen1\x_decorado#
 zg#=almacen1\z_decorado#
 yg#=TerrainY(terreno1,almacen1\x_decorado#,EntityY(m),almacen1\
     z_decorado#)
 PositionEntity m,xg+factor_dispersion,yg-20,zg+factor_dispersion
 RotateEntity m,0,Rnd(0,360),0
 Factor_dispersion=Rnd(100,dispersion)
Next
```



8

Un ejemplo de error típico derivado de la colocación aleatoria de objetos.

"TerrainY". Para posicionar el almacén vamos a tener en cuenta el factor de dispersión. De esta manera, evitamos que queden demasiados juntos. Para llevar a cabo este sistema, solo tenemos que sumar a las coordenadas X y Z el factor de dispersión que guardamos en la variable "factor_dispersion". Este factor estará comprendido entre 100 y el parámetro "dispersion".

Para terminar, también aplicamos una rotación en el eje Y aleatoria entre 0 y 360 grados con el fin de ganar variedad en el conjunto.

Podemos añadir más parámetros a la función como escalado del modelo, color, efecto del material, etc. También es evidente que el control que tenemos de la situación de los objetos no es muy preciso, ya que se colocan aleatoriamente. Por lo tanto, es muy normal que parte de los objetos del decorado queden en el aire al coincidir con un desnivel. Sin embargo es posible corregir este error controlando la situación de cada vértice del modelo, aunque esto resulta más complicado de implementar. Otra forma de evitar esto es utilizar un sistema más controlado como utilizar zonas editadas previamente como veremos a continuación.

CARGANDO ZONAS EDITADAS

Volviendo a la función "crear_decorado()" del módulo "funcpantaudio.bb", nos situamos en la segunda opción de juego, en donde creamos la zona de combate a partir de los datos contenidos en el fichero que produce el editor de zonas de combate. El sistema es realmente sencillo. En primer lugar, cargamos el fichero con los datos:

```
Case 2: ; Zona de juego editada
  fichero=ReadFile("c:\zone of
  fighters\gfx\datos\
  "+zona_de_combate$)
...
```

Seguidamente, entramos en un bucle "While .. Wend" en el que vamos leyendo el fichero secuencialmente para obtener los datos de cada objeto colocado por el editor hasta que no encontremos ninguno más (Ver Fig. 4):

```
While Not Eof(fichero)
  objeto$=ReadString(fichero)
  null1=ReadByte(fichero)
  null2=ReadFloat(fichero)
  posx#=ReadFloat(fichero)
  posy#=ReadFloat(fichero)
  posz#=ReadFloat(fichero)
  roty#=ReadFloat(fichero)
  null3=ReadByte(fichero)
  ...
```

Entre los datos leídos encontramos las coordenadas de situación (posX, posX, posZ) y el ángulo de rotación (rotY). Además, obtenemos el nombre del objeto que corresponde (objeto$), lo que es necesario para saber qué modelo debemos crear y situar. De tal forma, entramos en una estructura "Select..Case" en la que preguntamos qué objeto crear (Ver Código 3).

El funcionamiento es bien fácil: sólo tenemos que crear y posicionar el objeto dependiendo de los datos obtenidos del fichero.

No olvidemos, después de leer y colocar todos los objetos contenidos en el fichero, cerrarlo con "CloseFile (fichero)".

Como hemos comprobado, implementar un sistema automático de creación de decorados es una tarea sencilla y con buenos resultados, lo que permite que el juego resulte diferente cada vez. Por otro lado, la posibilidad de reproducir exactamente una zona de combate a partir de un editor trasciende en un mayor atractivo para el jugador, el cual podrá participar de lleno en un nivel original y propio.

Código 3. Preguntamos qué objeto crear

```
...
Select objeto$
 Case "almacen1"
  almacen1.tipo_almacen1= New tipo_almacen1
  almacen1\entidad_decorado=CopyEntity(mesh_almacen1)
  EntityType almacen1\entidad_decorado, ENTIDAD_EDIFICIO
  almacen1\x_decorado#=posx#
  almacen1\z_decorado#=posz#
  PositionEntity almacen1\entidad_decorado,posx#,posy#,posz#
  RotateEntity almacen1\entidad_decorado,0,roty#,0
 Case "almacen2"
  almacen2.tipo_almacen2= New tipo_almacen2
  ...
...
```

En el próximo número...

... nos sumergiremos en el sistema de inteligencia artificial del juego.

Fabricando los elementos del juego (VI)

Terminaremos en este número de texturizar los demás elementos del decorado del juego. Pero para llevar a cabo nuestra misión es necesario conocer algunas técnicas para obtener texturas de calidad.

Aparte de dibujar manualmente cualquier textura con un programa de diseño podemos utilizar otras técnicas como el retoque fotográfico o la utilización de aplicaciones 3D. Si, por ejemplo, necesitamos la textura de una pared siempre se puede obtener una fotografía de una de verdad y posteriormente retocarla en Paint Shop Pro o cualquier otra aplicación para que se amolde a nuestras necesidades. Realmente, este sistema es sencillo y no requiere mucho esfuerzo. Además, se consiguen resultados muy buenos ya que la textura parecerá real. Puede ocurrir que unas buenas texturas fotográficas no queden demasiado bien en un modelo con poco polígonos o que el conjunto de estas texturas no esté acorde con el resto de elementos del juego. Aun así, siempre se obtienen resultados estupendos. La utilización de aplicaciones de modelado y renderizado 3D también resulta interesante para la obtención de texturas preiluminadas. En el texturizado del resto de modelos del decorado vamos a utilizar esta técnica utilizando el programa Bryce 5.0 (aunque cualquier versión es válida).

TEXTURIZANDO EL PUENTE

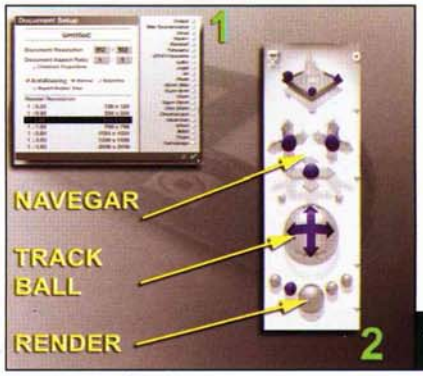
Para obtener la textura que configura el suelo y el techo del puente vamos a utilizar, como hemos dicho, Bryce 5. La idea es texturizar una primitiva como un cubo y aplicarle una textura; a continuación, orientamos el objeto hacia la cámara y grabamos una imagen del renderizado. Esta imagen contendrá la textura que necesitamos. Hay que tener en cuenta que si la textura que creemos con esta técnica va destinada a un tileado, necesitaremos hacerla cíclica con un programa de retoque para que al duplicarla no se note la unión entre los bordes. Sin embargo, en nuestro caso no utilizaremos tileado. Así que nos centraremos en nuestra textura individualmente. Una vez que hayamos obtenido la textura deseada en una imagen pasaremos a Paint Shop Pro para texturizar el puente a través de la plantilla.

OBTENIENDO LA PLANTILLA

Como sabemos, antes de texturizar necesitamos crear el mapeado UV del modelo. Para obtenerlo, utilizaremos el mismo procedimiento que usamos con los almacenes en el programa LithUnwrap. Cargamos el modelo, lo seleccionamos, aplicamos la opción *Tools / UV Mapping / Box*. Guardamos la plantilla con un tamaño de 512 x 512 en *File / Template / Save*.

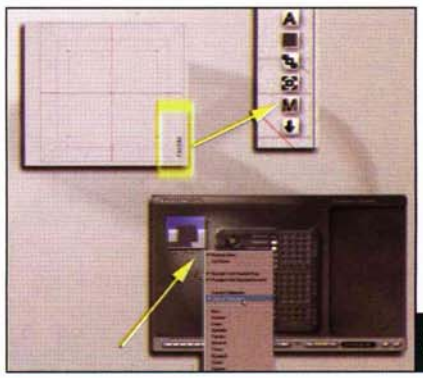
UTILIZANDO BRYCE

Instalemos Bryce 5 y ejecutemos el programa. No vamos a hacer una explicación exhaustiva de su funcionamiento, solo lo necesario para nuestras necesidades. Para empezar vamos a cam-



1 *Descripción del panel de navegación con las opciones "Navegar", "Track ball" y "Render".*

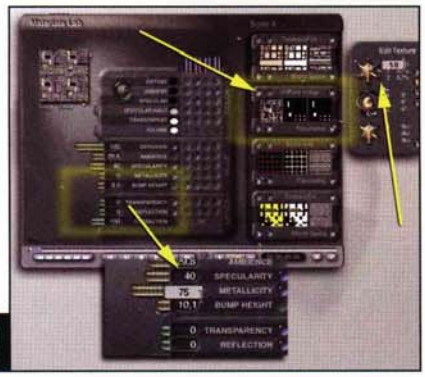
2 *Descripción de las herramientas de crear y editar: podemos especificar las luces, escalar o rotar la imagen, etc.*



3 *Para ir al laboratorio de materiales pulsamos sobre la letra "M" situada en la parte inferior derecha del objeto.*

4

En cada ventana que abrimos en el laboratorio de materiales disponemos de una ventana de previo.



5

En el laboratorio de materiales disponemos de todas las herramientas necesarias para modificar el material elegido.

NOTA

En la esquina superior izquierda del programa se encuentra una pequeña ventana de visualización que muestra un previo del renderizado del entorno actual.

biar el tamaño del documento. Elegimos la opción *File/Document Setup* y entramos en una ventana de diálogo en la que podemos cambiar la resolución de la pantalla. Deseleccionamos *Constrain Propotions* y en *Document Resolution* escribimos 512 x 512.

En el menú *create* situado en la parte superior, encontramos una serie de elementos que podemos crear. Elegimos el cubo y hacemos clic sobre él. Observamos cómo aparece en el entorno 3D sobre la cuadrícula. Para navegar por el entorno 3D encontramos a la izquierda una serie de iconos que describimos en la figura 1 / 2. Con la *Camara Track Ball* nos situamos en la línea de su cara frontal. (En la figura 2 se muestra una descripción de los iconos del menú *Edit*).

Posteriormente, en el menú *Edit* seleccionamos el icono de desplazamiento para situar el cubo a nuestra altura. Luego, escalamos el cubo con la herramienta de escalado hasta que cubra toda nuestra vista.

Ya casi estamos preparados para entrar en el editor de materiales. Como comentábamos, la idea es texturizar este cubo y el resultado grabarlo en una imagen. Pero antes, vamos a orientar la luz del sol para iluminar la cara del cubo que tenemos enfrente. Entramos en el menú *Sky & Fog*. A la derecha vemos una esfera que indica la posición del sol. Con el ratón situamos el sol en la parte inferior. Podemos ver cómo en el previo se ilumina la imagen. Ahora ya estamos preparados. Para entrar en el editor de materiales pulsamos en el icono con la letra "M" situado en la parte inferior derecha del cubo. Una vez dentro, podemos preparar la ventana de previo situada en la parte superior izquierda. Abrimos la lista de opciones de visualización pulsando en la flechita situada al lado de "paste"y elegimos *Actual Selection*. Entonces se mostrará la escena con nuestro cubo.

Orientamos la cámara del previo hasta situarla enfrente del cubo. Nos acercamos a él hasta que cubra toda la pantalla. A continuación seleccionamos el material que asignaremos al cubo. Para ello, pulsamos en la flechita situada en parte superior derecha del previo (Ver Fig. 3).

Entraremos en otra ventana con todos los tipos de materiales disponibles. Observamos que, de nuevo, disponemos de una ventana de previo. Vamos a elegir un material para la superficie del puente. Elegimos la clase *Miscellaneous* y la textura *Urban Dwelling* (Ver Fig. 4).

En el previo podemos ver nuestro cubo con la textura. Es importante tener el cubo totalmente perpendicular a la vista para así obtener una textura sin perspectiva. Una vez aceptada, volvemos al *Materials Lab*, en donde trabajaremos la apariencia de nuestra textura. Podemos observar a la derecha que el dibujo de la textura está formada por otras cuatro texturas. Cada una de ellas puede ser editada. Por el momento, solo vamos a modificar el tamaño de la segunda. Para ello, pulsamos sobre la pequeña esfera de la esquina superior izquierda para abrir el editor. En los valores X, Y y Z del factor de escala hay que escribir 30 para los tres ejes haciendo clic sobre los números. Para terminar, vamos a añadir un poco de efecto metálico y de altura (bump), asignando el valor 75 en *Metallicity*, 10 en *Bump Height* y aumentemos *Reflection* y *Refraction*. ¡Listo! Pulsamos en "aceptar" (esquina inferior derecha) y volvemos al entorno 3D (Ver Fig. 5).

Podemos ver entonces cómo en el previo se muestra una imagen de nuestro cubo ya texturizado. Para renderizar la escena solo tenemos que pulsar sobre la esfera más grande situada debajo de los iconos de navegación. Sin embargo, vamos a aprovechar las cualidades de iluminación que un programa de estas características nos ofrece. Ahora vemos que la textura resulta con un aspecto demasiado plano. Así que cambiemos el color y la posición del sol para obtener algún tipo de sombreado. Para cambiar el color del ambiente, nos situamos en *Sky & Fog* y pulsamos sobre el cuadradito situado debajo de la esfera de posición del sol. Elegimos un color celeste y luego cambiamos la posición del sol para que la luz incida lateralmente (Ver Fig. 6).

Una vez comprobada la textura pasamos a grabarla en disco. Para realizar este proceso debemos elegir la opción *Render to Disk...* del menú *File* (la barra del menú principal aparece cuando subimos el puntero del ratón hacia la parte superior de la pantalla). El programa nos pedirá la resolución de salida (queda como está) y a continuación el nombre del archivo. Elegimos el formato *.bmp* y guardamos.

Aprovechando la escena vamos a obtener la textura que nos servirá para los soportes curvos del techo del puente, la cual tiene aspecto de piedra. Pulsamos "ESC" y entramos de nuevo en el laboratorio de materiales. Lo único que tenemos que hacer es elegir una textura pétrea de la librería. Por ejemplo, "Arizona" de "Planes&Terrains" y escalarla al 30 %.

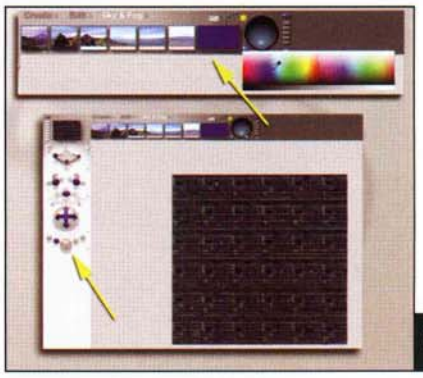
COMPLETANDO LA TEXTURA EN PAINT SHOP PRO

Para completar la textura en Paint Shop Pro emplearemos la misma técnica utilizada para los almacenes. Creamos un documento nuevo de 512 x 512 y cargamos la plantilla con el mapeado UV del puente. Pasemos la imagen de la plantilla a la nueva que hemos creado para tener una capa de referencia. Reduzcamos la opacidad de la capa para volverla semitransparente. Ahora carguemos la primera de las texturas que creamos con Bryce y la pasamos a la imagen principal. Colocamos la nueva capa por debajo de la capa de la plantilla. Seguidamente, con la herramienta de transformación ajustamos su tamaño a la base del puente, el cual corresponde al rectángulo mayor de la segunda vista de la plantilla. Luego, copiamos la capa para cubrir el rectángulo mayor de la quinta vista correspondiente al techo y a la parte superior del suelo. Hagamos lo mismo para el resto de las vistas para cubrir toda la base. No es la forma más correcta ya que, en cada vista, la textura adquiere una apariencia diferente, pero por el momento servirá (Ver Fig. 7).

Recordemos que el techo forma un rectángulo un poco más ancho que la base, así que tenemos que crear dos capas más para adaptarlas a estos rectángulos.

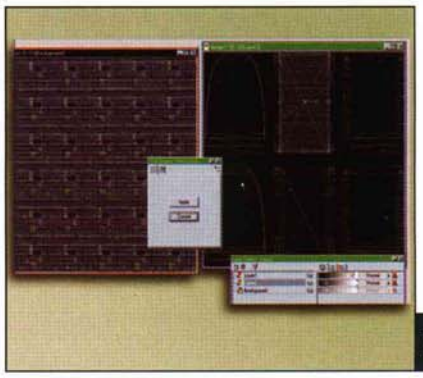
Para los soportes del puente utilizamos la segunda textura que creamos con Bryce. El procedimiento es el mismo. Cargamos la textura y la pasamos a la imagen principal. A continuación, la ajustamos a los soportes de la plantilla. No olvidemos situar la nueva capa por debajo de la capa de la plantilla. El procedimiento es el mismo: duplicar y situar según marque la plantilla (Ver Fig. 8).

Podemos utilizar la herramienta de borrar para ir perfilando aún más las texturas



6

Modificando la luz ambiental y el color podemos modificar el aspecto de la textura.

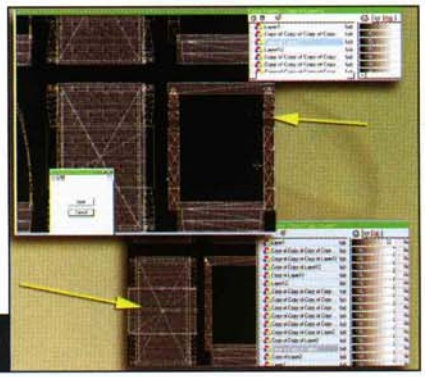


7

Una vez obtenida la textura, pasamos a ajustarla en la plantilla.

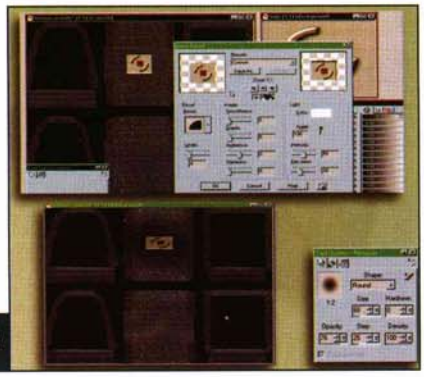
NOTA

Podemos mover y orientar la escena del previo haciendo clic y moviendo el ratón. Para alejar o acercar la cámara dejamos pulsado la tecla "Ctrl" mientras movemos con el ratón. Para rotar la escena con la tecla "Shift" y para desplazar la cámara con la tecla "ESPACIO". Para volver a la vista inicial, pulsar la tecla "Alt".



8

Debemos cubrir toda la plantilla haciendo copias de las capas y ajustando con la herramienta de transformación.



9

Después de colocar la placa del logo debemos retocar la iluminación de la textura para proporcionarle volumen al modelo.

10

La textura del cielo es sencilla de obtener utilizando Bryce.

(de forma rectangular) a las siluetas curvas. Ya solo nos queda colocar el logotipo del juego en el suelo del puente. Aprovechemos el que diseñamos para los almacenes. Lo cargamos y añadimos a la textura principal para aplicarle un efecto de relieve para que parezca que está incrustado en el metal. Ahora debemos retocar la textura con toques de iluminación para diferencia un poco las partes del puente. Pero antes, vamos a unir todas las capas en una sola imagen con la opción *Merge All /Flatten* en el menú que aparece al hacer clic con el botón derecho sobre cualquiera de las capas. Recordemos que hay que eliminar primero la capa de la plantilla, ya que no la necesitamos. Elegimos la herramienta *Retouch* en el modo *Darken RGB* y pintamos en las vistas 1, 3, 4 y 6 las separaciones de las tres partes que forman la base del puente. Esto dará más volumen al modelo (Ver Fig. 9).

Luego hagamos unas líneas longitudinales a lo largo de los soportes del puente. También podemos crear algunas partes más oscuras y claras en el suelo del puente para simular sombras debido a la presencia del techo. Todos los retoques posibles son siempre bienvenidos. Para finalizar, siempre podemos ver el resultado inmediatamente por medio de LithUnwrap.

TEXTURIZANDO LAS ROCAS, ÁRBOLES Y EL CIELO

Como habíamos comentado, las texturas de las rocas y árboles las vamos a realizar utilizando la técnica empleada con Bryce. Para las rocas solo tenemos que elegir la textura pétrea que más nos guste y para los árboles podemos utilizar, si usamos Bryce 5, la nueva librería de texturas para troncos de árboles *Trunks*. Para la realización del cielo vamos a utilizar el renderizado que Bryce tiene de las nubes. Dejemos el entorno sin ningún objeto y apuntamos con la cámara hacia arriba. Nuestra idea es obtener una textura difusa con muchos colores. Así que modifiquemos la estructura de las nubes y tomemos una instantánea. Para modificar el cielo, encontramos todos los parámetros en *Sky & Fog*. Pulsando en la flechita situada a la derecha de *Sky&Fog* entramos en la librería de cielos deBryce. Si usamos la versión 5, una buena opción es el cielo *Middle Of The Sun*. En este apartado podemos apreciar seis ventanitas, cada una con un dibujo de un paisaje. Cada una de ellas representa una opción de modificación del ambiente: sombras, niebla, altura o cantidad de las nubes, etc. La gráfica situada a la izquierda de la esfera de situación del sol representa la frecuencia de aparición de nubes. Haciendo clic y desplazando el ratón podemos modificar su valor. Jugando con todos estos parámetros podemos conseguir el cielo que queramos. Luego solo tenemos que renderizar a disco y ya tenemos nuestra textura. Aunque también podemos texturizar el cubo anterior utilizando la librería de nubes del laboratorio de materiales. Obtendremos texturas más rectas y lineales con este método.

Aprovechando que hemos entrado en el uso de Bryce, vamos a desarrollar el mapa de alturas de nuestro terreno con esta aplicación.

En el próximo número...

... fabricaremos el terreno con Bryce.

Nuestro primer tema musical con Anvil Studio (IV)

En esta entrega terminaremos de aprender la forma de editar las pistas MIDI en la sección *Compose*, modificando los eventos en el panel *Piano Roll*.

Además, entraremos en la creación y edición de pistas de muestras.

EDITANDO EN EL MODO *PIANO ROLL*

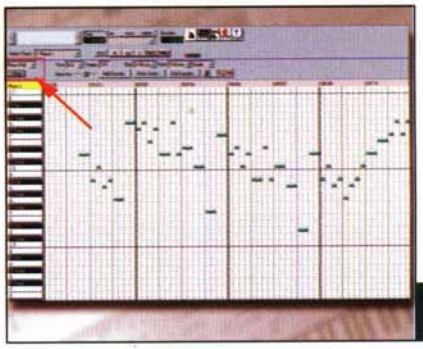
Otra opción de la que disponemos para editar nuestras pistas MIDI es el modo *Piano Roll*. En él se muestra de una forma gráfica cada una de las notas de la pista sobre una cuadrícula que representa las divisiones de cada compás. Antes de seguir, carguemos en el programa el tema "FugueGM" habitual. Seleccionamos la primera pista y pasamos a la sección *Compose*. Una vez allí, donde pone *Staff* elegimos *Piano Roll* de la lista. Observamos cómo en la cuadrícula se reparten una serie de rectángulos, los cuales representan cada nota. La longitud de cada rectángulo significa la duración de la nota. Y el tono dependerá de la posición vertical en la cuadrícula. De todas formas, a la izquierda de la ventana podemos ver un teclado que nos ayudará a tener una referencia del tono (Fig. 1).

EDITANDO LAS NOTAS

La forma de trabajar con las notas en este modo difiere un poco del modo *Staff*. Para cambiar el tono de una nota (rectángulo), es decir, desplazarla hacia arriba o hacia abajo, nos colocamos sobre ella y la movemos con el ratón (es conveniente elegir una nota de bastante duración para practicar mejor). Si lo que queremos es desplazarla hacia delante o atrás en el compás, debemos situar el puntero del ratón al comienzo de la nota, entonces, éste cambiará de forma. Pasará de la cruz a una flecha de doble sentido. Sin dejar de hacer clic desplazamos el ratón hacia la dirección deseada. Vemos cómo al dejar de hacer clic la nota se adapta automáticamente a la división de la cuadrícula más cercana (cuantización) (Fig. 2).

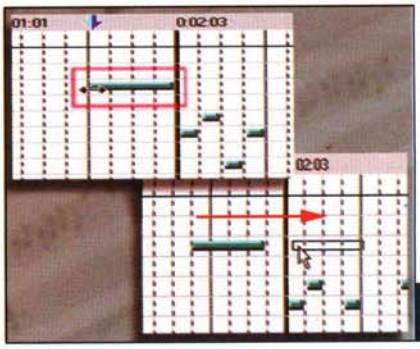
Para alargar o reducir la duración de la nota, simplemente nos situamos al final del rectángulo con el puntero del ratón; éste, de nuevo, cambiará su aspecto al de una flecha de doble sentido. Sin dejar de hacer clic desplazamos el ratón hacia la derecha o izquierda. En esta ocasión, al dejar de hacer clic, la nota se adaptará a la cuadrícula dependiendo de si tenemos la opción *Snap to Grid* activada o no a través del icono .

Para colocar una nueva nota, es suficiente con hacer clic en cualquier lugar de la cuadrícula. La nota nueva tendrá la duración definida en la casilla "Note". Por defecto adquiere un valor de 1 / 4; es decir, una parte de un 4 por 4. Pulsando con el botón derecho del ratón sobre cualquiera de las notas aparecerá un menú flotante en el que podemos cambiar sus propiedades (duración y volumen) o borrarla (Ver Fig. 3).



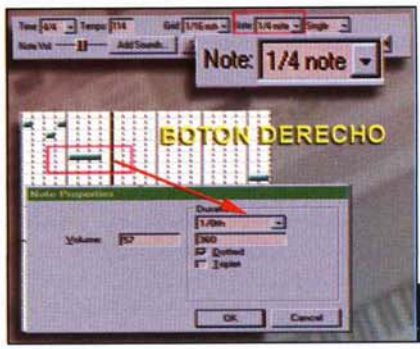
1

Modo Piano Roll de la sección Compose.



2

Procedimiento para desplazar una nota con el ratón en el tiempo.

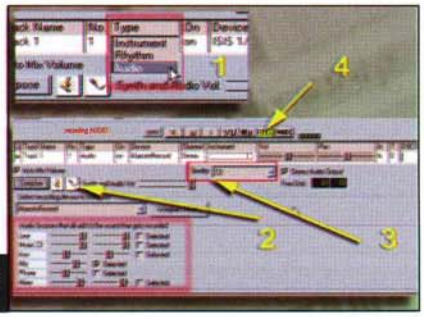


3

La duración de la nueva nota creada dependerá del valor elegido en la casilla "Note".

4

Para seleccionar grupos de notas debemos hacerlo en la línea de tiempo.



5

Pasos para grabar una pista de audio desde el micrófono.

CD	16 bit 44 Khz
Video Juego	8 bit 44 Khz
Fonógrafo	16 bit 22 Khz
Radio (FM)	16 bit 11 Khz
Radio	8 bit 22 Khz
Teléfono	8 bit 11 Khz
Radio antigua	8 bit 8 Khz

6

Esquema de las calidades de audio posibles en Anvil Studio.

TRUCO

Si queremos ver al mismo tiempo la sección *Compose* en *Piano Roll* y las pistas de *Mixer*, después de reducir la vista, pulsamos en el botón "Mixer" situado bajo *Piano Roll* y luego volvemos a la sección *Compose* pulsando en el botón "Compose".

EDITANDO GRUPOS DE NOTAS

Antes de seguir, vamos a reducir un poco la cuadrícula para poder observar mejor todas las operaciones de edición. Para ello, hacemos clic en el icono de la lupa con el signo menos .

Vamos a copiar un grupo de notas de un lugar a otro. Par seleccionar las notas debemos hacerlo en la línea de tiempo (Ver Fig. 4).

Por ejemplo, seleccionemos un compás entero. Desplazamos el ratón hacia la derecha sin dejar de hacer clic. Después, hacemos "Ctrl" + "C" para copiar.

Nos situamos dos o tres compases después haciendo clic en la línea de tiempo e insertaremos el compás copiado con la opción *Paste Insert* ("Mayús" + "Insert") en el menú *Edit*.
Observamos cómo el compás seleccionado al principio se inserta en el lugar elegido. Del mismo modo que seleccionamos un compás entero, podemos trabajar solo con un grupo de notas. Las opciones de copiado funcionan de igual manera que en el modo *Staff*; es decir, insertar, mezclar o solo un evento especial.

OTRAS POSIBILIDADES

Al igual que ocurre con las notas, podemos insertar un silencio situándonos en la línea de tiempo y pulsando en el icono . La duración del silencio dependerá de la opción elegida en la casilla "Note".

Hay una opción muy interesante en Anvil Studio que es la creación y gestión de Loops (bucles) de una manera realmente completa. Pèro este tema lo dejaremos para una próxima entrega debido a que haremos un estudio detallado de su funcionamiento y utilización.

PISTAS DE AUDIO

Otra de las características que Anvil Studio admite es la posibilidad de crear y editar pistas de audio. Inicialmente, solo permite una sola pista de audio (mono o estéreo). Si se quiere más, es necesario adquirir el accesorio *Anvil Studio MultiAudio*. De todas formas, con una sola es suficiente para nuestros propósitos.

GRABANDO UNA PISTA DE AUDIO A TRAVÉS DEL MICRÓFONO

Antes de empezar, nos situamos en *Mixer* y creamos una nueva canción en *File / New*. Cambiamos el tipo de pista "instrument" (por defecto) por una de audio. Seleccionamos la tarjeta de sonido pulsando en el icono y configuramos el volumen adecuado del micrófono. Podemos, además, seleccionar si la pista de audio es mono o estéreo en la casilla "Channel".
Seguidamente, solo tenemos que pulsar en el icono de grabación "REC". A continuación, el programa pedirá el nombre del nuevo fichero de audio que se va a crear y su ubicación. Una vez proporcionado, el programa empezará a grabar (Fig. 5).

La calidad de la grabación la podemos elegir en la casilla "Quality". En la figura 6 se muestra un esquema con los distintos tipos de calidades soportados por el programa.

GRABANDO UNA PISTA DE AUDIO DESDE EL CD

Si lo que deseamos es importar una pista de CD a la pista de audio, simplemente elegimos la calidad "CD"y pulsamos en el botón "VU". Después de hacer play en el CD pulsamos en "REC".

En el próximo número...

... terminaremos con el sistema de manejo de pistas de audio.

Teclado, ratón y dispositivos de juegos

En todo videojuego o aplicación multimedia es fundamental la comunicación con el usuario.

La interactividad es la esencia del videojuego. Sin ella, el concepto de juego desaparecería. No solo es primordial una comunicación entre ordenador y humano, sino que ésta debe ser rápida, fluida y cómoda. Como todo lenguaje de programación, Blitz3D permite el manejo del teclado, ratón y otros dispositivos de entrada de datos manual como un gamepad o un joystick. En este número trataremos de explicar la utilización de estos elementos.

FUNCIONES DE TECLADO

Hasta que no se perfeccione y estandarice el reconocimiento de voz, el teclado será la única herramienta para introducir caracteres en el ordenador. La manera más conocida de satisfacer las peticiones de un programa es a través de la instrucción "Input":

```
Input (prompt$)
```

El "prompt" simplemente te indica que el ordenador está listo para recibir datos desde el teclado. Se mantiene a la espera mostrando un cursor, sin embargo, es posible añadir una frase además del cursor. Para utilizar "Input" es necesario asignar la operación directamente a una variable. Dependiendo del tipo de variable, "Input" automáticamente, sabrá qué tipo de datos se va a introducir. Por ejemplo, para obtener una frase:

```
Frase$= Input ("Escriba una
frase "): Print Frase$
```

Para escribir un número:

```
Numero= Input ("Escriba un
valor ")
```

Es importante saber que si no se está usando ningún modo gráfico, el "prompt" aparecerá en la pantalla directamente. Si por el contrario utilizamos modo gráfico, se mostrará en el búfer de dibujo actual.

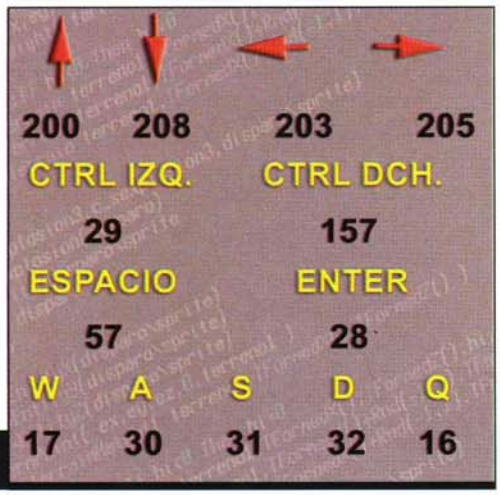
Otra forma de obtener datos desde el teclado es utilizando "GetKey()". La diferencia con "Input" es notable. Mientras que "Input" detiene la ejecución del programa a espera de datos y la aceptación con "Return", "GetKey()" no necesita hacerlo. Sin embargo, solo obtiene un carácter a la vez y solo su valor ASCII, el cual será almacenado en una variable.

Por lo tanto, esta variable debe ser de tipo numérico: Caracter = GetKey()

```
KeyHit (Cod. Tecla)

While Not KeyHit (1)
  PROCESOS
Wend

KeyDown (Cod. Tecla)

If KeyDown (205)
  MOVER A LA DCHA.
Endif
```

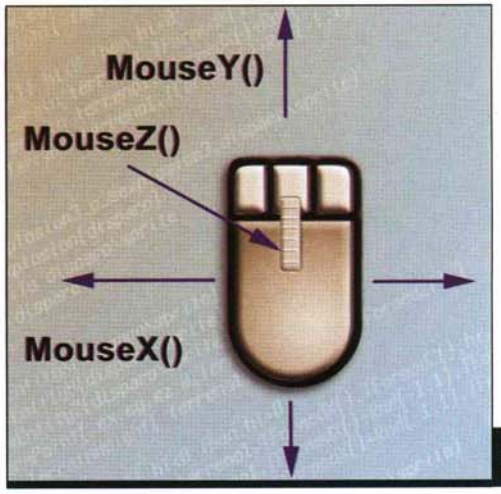
2

Usos más comunes de las instrucciones "KeyHit()" y "KeyDown()".



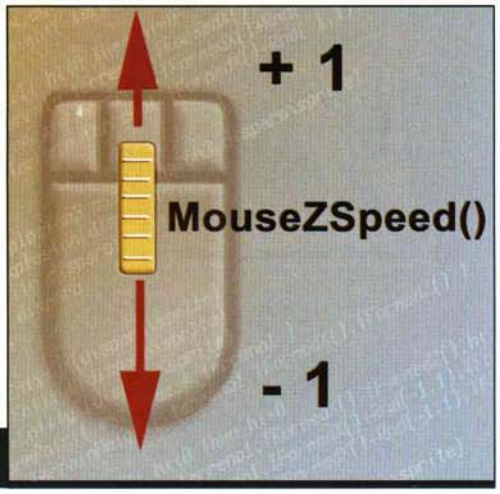
1

Esquema de los códigos de teclas más usados.



3

Funcionamiento de "MouseX()", "MouseY()" y "MouseZ()".



4

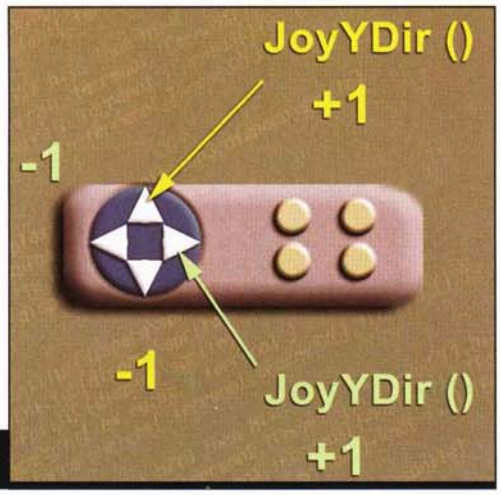
Funcionamiento de "MouseZSpeed()".

```
While Not numero
    Numero=GetKey()
Wend
Print numero
```

En este pequeño ejemplo, si pulsamos la tecla "A" el programa imprimirá "194" que corresponde a su valor en la tabla ASCII.

Sin embargo, si queremos detectar más de una tecla a la vez o una tecla especial; por ejemplo, "Ctrl + A"o simplemente "Crtl", necesitaremos otro tipo de instrucción. Blitz3D nos soluciona este problema ofreciéndonos los comandos "KeyHit", "KeyDown" y "WaitKey()".

El primero de ellos:



5

Las instrucciones "JoyXDir()" y "JoyYDir()" son ideales para el manejo de joysticks digitales.

```
KeyHit (Código de
la tecla pulsada)
```

Nos informa de cuántas veces se ha pulsado una determinada tecla. Es ideal, por ejemplo, para detectar la tecla "Escape" para salir de un bucle.

```
While Not KeyHit(1)
    PROCESOS
Wend
End
```

El "Código de la tecla pulsada" es un valor que corresponde a una tecla. En el CD se encuentra la lista de cada uno de estos códigos y la tecla a que corresponde "CODIGOS_TECLADO".

A veces necesitaremos saber qué tecla está siendo pulsada. Para ello utilizaremos el segundo de los comandos que Blitz3D nos proporciona:

```
KeyDown (Código de la tecla
pulsada)
```

Este comando es el que debemos utilizar si queremos detectar varias teclas a la vez, como por ejemplo, "Ctrl+A" , "Shift + Ctrl + V" o los cursores:

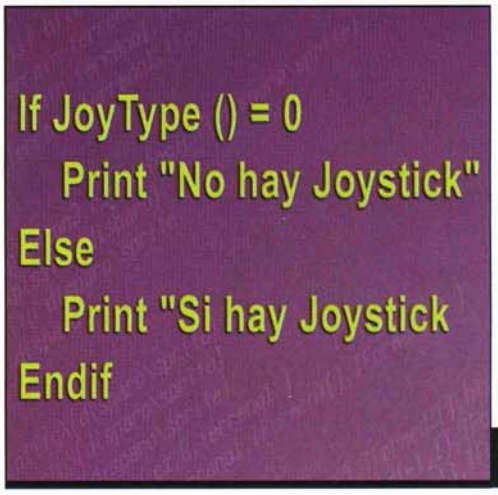
```
...
If KeyDown(29) And
KeyDown(194) Then Print
"Se ha pulsado Ctrl + A"
...
```

Este comando devuelve el valor 0 si no se ha pulsado la tecla y un 1 en caso contrario. Por lo tanto, la anterior sentencia es equivalente a esta otra:

```
If KeyDown(29)=1 And
KeyDown(194)=1 Then Print
"Se ha pulsado Ctrl + A"
```

La última instrucción de que disponemos es "WaitKey()" y se usa generalmente sola y cuando queremos parar la ejecución del programa hasta que se pulse cualquier tecla. También se puede usar para asignar el código ASCII de una tecla a una variable.

Cada vez que pulsamos las teclas en operaciones de input van creando una cola de espera en la memoria. Para borrar esta porción de memoria se utiliza "FlushKeys".



6

Código para la detectar si hay joystick conectado.

FUNCIONES DE RATÓN

Blitz3D también permite controlar el manejo de un ratón de hasta 3 botones y rueda deslizadora.

Al igual que ocurre con los comandos para detectar la pulsación del teclado "KeyHit" y "KeyDown", también los disponemos para el ratón: "MouseHit" y "MouseDown":

```
MouseHit (Nº del botón)
```

Este comando cuenta las veces que se ha hecho clic con el ratón desde la última vez que se llama. El parámetro "Nº del botón" puede valer 1, 2 ó 3 dependiendo de qué botón se ha pulsado:

- 1 ➡ Botón izquierdo
- 2 ➡ Botón derecho
- 3 ➡ Botón central

Eje_X = JoyX ()
Eje_Y = JoyY ()
Eje_Z = JoyZ ()

Joystick Analógico

7

Instrucciones correctas para detectar el movimiento de un Joystick Analógico.

Eje_X = JoyXDir ()
Eje_Y = JoyYDir ()
Eje_Z = JoyZDir ()

Joystick Digital

8

Instrucciones correctas para detectar el movimiento de un Joystick Digital.

```
While Not MouseHit(1)
    PROCESOS
Wend
```

En este ejemplo, hasta que no se pulsa el botón izquierdo no se sale del bucle.

Para detectar si se ha pulsado contamos con el comando "MouseDown":

```
MouseDown (Nº del Botón)
If MouseDown(1) = 1  Print "Se
ha pulsado el botón izquierdo"
```

En ocasiones necesitaremos saber qué botón se ha pulsado. Para ello, disponemos de "GetMouse":

```
Boton = GetMouse()
While Not KeyHit(1)
  Boton=GetMouse()
  Select Boton
  Case 1: Print "Se ha pulsado
  el botón izquierdo"
  Case 2: Print "Se ha pulsado
  el botón derecho"
  Case 3: Print "Se ha pulsado
  el botón central"
  End Select
Wend
```

Si no se pulsa ninguno devolverá el valor 0.

El equivalente a "WaitKey()" lo encontramos en "WaitMouse()". Su función es exactamente la misma. Supongamos que después de realizar operaciones con los botones del ratón hacemos una petición al usuario en el que se pide que se pulse un botón, por ejemplo, para salir con "WaitMouse()". Debemos entonces borrar la cola de pulsaciones anterior. Por lo tanto, utilizaremos "FlushMouse".

Por defecto, en cualquier aplicación, Blitz3D está continuamente detectando el ratón; es decir, se encuentra en todo momento en pantalla. Lo que ocurre es que, en modo de pantalla completa, se mantiene invisible. Para verlo, debemos asignar a sus coordenadas un gráfico, que, por ejemplo, represente un puntero. Para saber entonces la posición exacta del ratón disponemos de las instrucciones "MouseX()","MouseY()".

```
While Not KeyHit(1)
  ...
  Drawimage puntero, MouseX(),
  MouseY()
  ...
Wend
```

"MouseX()" nos devuelve la coordenada X de la posición del ratón y "MouseY()" la coordenada Y.

Podemos necesitar, entonces, colocar el puntero en un determinado lugar de la pantalla. Para ello, podemos utilizar "MoveMouse":

```
MouseMove Coord. X, Coord. Y
```

Blitz3D nos permite, además, detectar el deslizamiento de la rueda que algunos modelos de ratón llevan incorporado con la instrucción "MouseZ()". El valor que devuelve "MouseZ()" se incrementará cuando se mueve la rueda hacia delante y decrementará cuando se mueva hacia atrás.

```
While Not KeyHit(1)
  Posicion = MouseZ()
  Print "Posición de la rueda
  "+ Posicion
Wend
```

No queda aquí el control que podemos hacer del ratón. También es posible, incluso, detectar los cambios de posición del ratón entre llamadas; es decir, saber dónde estuvo el ratón antes y dónde está ahora. Para obtener esta información tenemos las instrucciones "MouseXSpeed" y

If KeyDown (29) and
KeyDown (30)
Print "CTRL + A"
Endif

9

Ejemplo de cómo utilizar "KeyDown" para detectar una combinación de teclas.

```
While Not Salir
  Salir=GetKey()
  Color Rnd(100,255),40,50
  Rect Rnd(10,500),Rnd(10,400),40,50
Wend
```

10

Utilización de "GetKey" para salir de un bucle.

```
While Not KeyHit (1)
  Color Rnd(100,255),40,50
  Rect Rnd(10,500),Rnd(10,400),40,50
Wend
```

11

Otra forma de salir de un bucle al pulsar la tecla "escape". Esta vez con "KeyHit".

"MouseYSpeed". Podemos ver cómo funcionan estos comandos en el ejemplo "ejemplo1.bb".

Cuando se hace una primera llamada a la función, ésta graba la situación del ratón. Al llamarse por segunda vez, devuelve el desplazamiento del ratón entre las dos llamadas.

Con la instrucción "MouseZSpeed()" averiguaremos en qué dirección se ha girado la rueda del ratón.

```
Direccion = MouseZSpeed()
Direccion ➡ -1 hacia delante
Direccion ➡  1 hacia atrás
Direccion ➡  0 no se ha movido
```

DISPOSITIVOS PARA JUEGOS

El total control de dispositivos para juegos como joystick o gamepad está presente en Blitz3D a través de numerosas funciones. Para empezar, es importante saber si hay algún dispositivo conectado ordenador, y si es así, de qué tipo es. Para averiguar esta información disponemos de la instrucción "JoyType":

```
JoyType ([Puerto])
```

Esta función devuelve un 0 si no hay nada conectado, un 1 si el joystick es digital y un 2 si es analógico. La opción "Puerto" le indica a Blitz3D que chequee solo en el puerto de conexión especificado. Al igual que ocurre con las funciones del ratón para detectar la pulsación de los botones, disponemos de las mismas funciones para el joystick: "JoyDown", "JoyHit" y "GetJoy":

```
JoyHit (Botón [,Puerto de conexión])
```

Devuelve el número de veces que se ha pulsado el botón especificado del joystick.

```
JoyDown (Botón [,Puerto de conexión])
```

Detecta cuándo un botón está siendo pulsado. Blitz3D automáticamente puede detectar todos los botones del joystick conectado, ya tenga 3 o 6. Y por último: GetJoy ([Puerto]) devuelve el botón que ha sido pulsado. Para controlar el desplazamiento y dirección de los ejes del joystick ya sea analógico (palanca) o digital (gamepad, botones) disponemos de comandos extremadamente sensibles.

Con "JoyX","JoyY" y "JoyZ".

```
JoyX ([Puerto de conexión])
```

y

```
JoyY ([Puerto de conexión])
```

Devuelven los valores de los ejes X e Y del joystick o gamepad. El valor devuelto es del tipo *float* y va desde -1 hasta 1. La función "JoyZ" es similar a "JoyX" a diferencia de que los valores devueltos representan el máximo (1) o el mínimo (-1) en el desplazamiento.

Por último, podemos saber en qué dirección se han movido los ejes X, Y y Z del joystick mediante los comandos "JoyXDir", "JoyYDir" y "JoyZDir".

Si el joystick se mueve hacia la izquierda "JoyXDir" devolverá el valor entero -1 y si lo hace hacia la derecha, devolverá un 1. Si se mueve hacia arriba "JoyYDir" o JoyZDir" devolverá un -1 y hacia abajo un 1.

Para finalizar, podemos también borrar la cola de botones pulsados que se graba en memoria utilizando la instrucción "FlushJoy".

El uso que le demos a estas funciones dependerá, como es lógico, de nuestras necesidades. Pero hay que tener siempre en cuenta, por ejemplo, en un juego, que la velocidad de respuesta de nuestro programa en las decisiones del usuario es primordial para la jugabilidad. Es preciso, entonces, utilizar los comandos idóneos según las necesidades, así que es importantísimo estudiar todas las posibilidades que Blitz3D nos brinda.

NOTA

Si se quiere detectar joysticks digitales como gamepads es conveniente el uso de las funciones "JoyXDir", "JoyYDir" y "JoyZDir".

En el próximo número...

... entraremos de lleno en el manejo de ficheros.

Grabación y reproducción del movimiento de la cámara

Un elemento muy importante en todo videojuego, y cada vez más utilizado, es la introducción de cinemáticas (animación) en las partidas utilizando el propio motor gráfico del juego.

Estas animaciones pueden ser de muchos tipos dependiendo de la complejidad de las escenas que se quieren reproducir. Desde simples movimientos de cámara hasta complejos trozos de películas mostrando personajes, acción y narrativa. En este tutorial vamos a desarrollar la más sencilla de ellas, el movimiento de la cámara. Aun así, resulta tremendamente útil para un juego o cualquier otra aplicación multimedia, bien para situar la posición de una zona determinada o para realizar un vuelo por una superficie.

SISTEMA DE GRABACIÓN UTILIZANDO MATRICES

La idea básica del proceso se basa en la grabación de la posición y rotación de la cámara al mismo tiempo que es movida por el usuario. El almacenamiento de estos datos se puede hacer de varias formas. Una de ellas podría ser utilizando una matriz para cada dato. Este array se dimensionaría previamente antes del proceso de grabación con una cantidad de posiciones determinada:

```
Duracion=1000 ; Nº Posiciones
Dim PosX#(Duracion),
PosY#(Duracion),
PosZ#(Duracion)
Dim Pitch#(Duracion),
Yaw#(Duracion)
...
```

Bien, este sistema podría funcionar perfectamente. Simplemente, utilizaríamos un contador para desplazar el índice en cada dibujado de la escena y almacenaríamos los datos correspondientes:

```
While Not KeyDown(1)

  MOVER CAMARA

  PosX#(Indice) =EntityX(Camara)
  PosY#(Indice) =EntityY(Camara)
  PosZ#(Indice) =EntityZ(Camara)
  Pitch#(Indice) =EntityPitch(Camara)
  Yaw#(Indice) =EntityYaw(Camara)

  Indice=Indice+1
  If Indice=Duracion FIN GRABACION

  RenderWorld
  Flip
Wend
```

Sin embargo, en nuestro tutorial vamos a utilizar otro procedimiento más elegante, rápido y efectivo utilizando bancos de memoria.

BANCOS DE MEMORIA

El uso de matrices es más lento y consume memoria. Para nuestro proceso de grabación necesitamos un sistema más seguro y que utilice la memoria justa. Para ello, Blitz3D permite escribir y leer directamente en memoria utilizando bancos de memoria.

Para crear un banco de memoria se utiliza el comando "CreateBank()":

```
CreateBank (Tamaño)
```

El tamaño viene determinado por el tipo de datos que se van a almacenar. En la figura 1 se muestra una tabla con la memoria ocupada por cada uno de ellos.

Tipo	Memoria Ocupada	Rango
Byte	1 Byte	0 - 255
Short	2 Bytes	0 - 65535
Int	4 Bytes	-2147483647 2147483647
Float	4 Bytes	$-3{,}4 \times 10^{38}$ $3{,}4 \times 10^{38}$

1

Tabla de la memoria en bytes que ocupa cada tipo de datos y su correspondiente rango de valor.

Para escribir en la memoria disponemos de una instrucción para cada tipo de datos (Ver Fig. 2):

```
PokeByte Banco, Desplazamiento,
Valor
```

Esta instrucción almacena un byte de valor (Valor) en el banco de memoria "Banco". El desplazamiento viene determinado por la cantidad de memoria que necesita; es decir, para un valor "Short" el desplazamiento irá de dos en dos, ya que toma dos bytes para almacenar cada dato. Las siguientes instrucciones son: *PokeShort, PokeInt* y *PokeFloat* (Ver Fig. 2).

DATO	INSTRUCCIÓN
Byte	PokeByte
Short	PokeShort
Int	PokeInt
Float	PokeFloat

2

Instrucciones disponibles para escribir en memoria los distintos tipos de datos.

Para leer un valor almacenado en memoria disponemos de: *PeekByte, PeekShort, PokeInt, PeekFloat*. Por ejemplo (Ver. Fig. 3):

```
PeekByte (Banco, Desplazamiento)
```

SISTEMA DE GRABACIÓN UTILIZANDO BANCOS DE MEMORIA

Vamos a aprovechar el ejemplo del mapeado cúbico del tutorial número 12 para probar nuestro sistema de grabación y reproducción contenido en el CD con el nombre "grabación.bb". Le añadiremos estos procedimientos en el bucle principal.

Antes de entrar en el bucle principal vamos a reservar memoria para los datos con "CreateBank" (Ver Código 1).

Necesitaremos almacenar valores de tipo *float* para las coordenadas y ángulos, así que necesitaremos 4 posiciones de memoria para cada dato. Un cálculo aproximado nos lleva a que un segundo de animación ocupa 38 posiciones de memoria, por lo tanto necesitaremos 4 veces más. Seguidamente, creamos un banco para cada dato. Ya dentro del bucle principal, utilizaremos la variable "Offset" para realizar el desplazamiento en cada fotograma. Básicamente, el funcionamiento se reduce a ir grabando en memoria la posición y ángulos actuales de la cámara una vez pulsado el botón izquierdo del ratón (Ver Fig. 4). (Ver Código 2).

Para reproducirlos, solamente tenemos que situarnos al principio de la memoria grabada poniendo a cero la variable "Offset" y leer con "PeekFloat". (Ver Código 3)

Este sistema puede ser mejorado en muchos aspectos como la suavidad de rotación o la grabación en un fichero de los datos almacenados para su posterior uso. Todo depende de nuestras necesidades.

DATO	INSTRUCCIÓN
Byte	PeekByte
Short	PeekShort
Int	PeekInt
Float	PeekFloat

3

Instrucciones disponibles para leer de memoria los distintos tipos de datos.

Código 1. Reservamos memoria para los datos

```
Tiempo=Input("¿ DURACION DE LA GRABACION EN SEGUNDOS ? ")
Posiciones=Tiempo*38  ; Nº posiciones 1 seg. aprox. 38 posiciones
Memoria=Posiciones*4  ; 4 bytes por posición
BankMemX=CreateBank(Memoria)        ; Memoria para posición X
BankMemY=CreateBank(Memoria)        ; Memoria para posición Y
BankMemZ=CreateBank(Memoria)        ; Memoria para posición Y
BankMemPitch=CreateBank(Memoria)    ; Memoria para Pitch
BankMemYaw=CreateBank(Memoria)      ; Memoria para Yaw
```

Código 2. Vamos grabando en memoria

```
If Offset>Memoria-8    SW_Grab=False
If SW_Grab
    Offset=Offset+4
    PokeFloat BankMemX,Offset,EntityX(Camara)
    PokeFloat BankMemY,Offset,EntityY(Camara)
    PokeFloat BankMemZ,Offset,EntityZ(Camara)
    PokeFloat BankMemPitch,Offset,EntityPitch(Camara)
    PokeFloat BankMemYaw,Offset,EntityYaw(Camara)
EndIf
```

Código 3. Ponemos a cero la variable "Offset"

```
If SW_Play
  X#=PeekFloat(BankMemX,Offset)
  Y#=PeekFloat(BankMemY,Offset)
  Z#=PeekFloat(BankMemZ,Offset)
  RX#=PeekFloat(BankMemPitch,Offset)
  RY#=PeekFloat(BankMemYaw,Offset)
  PositionEntity Camara,X,Y,Z
  RotateEntity Camara,RX#,RY#,0
  Offset=Offset+4
  If Offset>Memoria-8 SW_Play=False
  If PeekFloat(BankMemX,Offset)=.0
     TurnEntity Camara,IPitch,0,0
     RotateEntity Camara,0,IYaw,0
     PositionEntity Camara,IPosX,IPosY,IPosZ
     SW_PLay=False
  EndIf
EndIf
```

NOTA

El sistema utilizado en este tutorial permite grabar hasta 100 horas de movimientos de cámara de una sola vez ocupando solo 9 MB de RAM.

En el próximo número...

... empezaremos una serie dedicada a la realización de un vídeo de presentación para nuestro juego.

Aventuras gráficas

Las aventuras gráficas son un género muy especial. Es como estar dentro de un libro de aventuras y participar en su argumento. Siempre ha sido ideal para el disfrute de toda la familia porque alimenta la imaginación y carece prácticamente de violencia gratuita. En esta entrega recorreremos un poco su historia y descubriremos el avance que ha sufrido en la última década.

COMIENZOS Y EVOLUCIÓN

Para conocer el comienzo de las aventuras gráficas debemos remontarnos al nacimiento de la aventura propiamente dicha. En los comienzos de los juegos para ordenador, el género de aventuras solo existía de forma conversacional, sin gráficos.

AVENTURAS CONVERSACIONALES

La primera aventura para ordenador fue *Collosal Cave Adventure* de Crowther y Woods en 1976. Estaba escrita en FORTRAN y no poseía gráficos algunos. Las diferentes situaciones eran explicadas en forma de texto en la pantalla y el jugador interactuaba por medio de acciones escritas utilizando verbos. Por ejemplo, para desplazarnos se utilizaban: "Subir","Norte","Dentro", etc. y para interactuar con los objetos se usaba el verbo más un sustantivo como: "Coger llave" o "Abrir puerta". Este sistema era muy simple pero eficaz y simulaba los libros interactivos de aventuras. Este tipo de juegos abundaba en los ordenadores de la época. Solo se necesitaba un teclado para jugarlos y conocimientos de inglés.

Posterior a *Adventure* apareció la serie *Zork* (traducida al castellano), *Planetfall* y *Leather Goddesses of Phobos*. Poco a poco empezaron a aparecer los gráficos y los juegos adquirieron un sistema híbrido en el que se seguían utilizando comandos escritos como interfaz de usuario y gráficos para describir las situaciones.

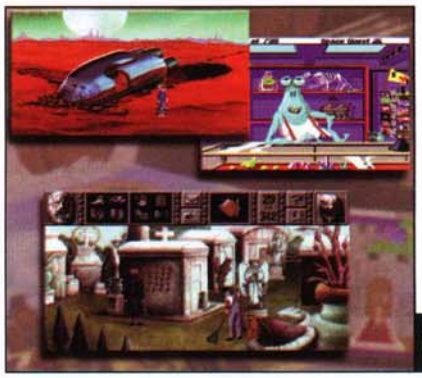
NACE LA AVENTURA GRÁFICA

La persona responsable de crear la primera aventura realmente gráfica fue *Roberta Williams* en el año 1980 cuando publicó para Apple II el juego *Mystery House*. Williams, además, dotaba a sus títulos de un carácter muy personal con historias complejas y absorbentes. Todavía los juegos de aventuras carecían de movimiento alguno. Pero de nuevo Williams junto a su marido deciden desarrollar esta nueva posibilidad. Fundan la empresa Sierra y publican la serie de aventuras más popular de todos los tiempos, *King´s Quest*, en 1984. Estos títulos seguían siendo híbridos pero los personajes del juego ya poseían movimiento.

Aprovechando las posibilidades gráficas del PC empezaron a aparecer aventuras de tipo híbrido con más movimientos y con pequeñas secuencias de animación como la *Leisure Suite Larry* (1987). También se versionaban para PC títulos de los 8 bits como *The Hobbit*. En España llegaban algunos títulos de la mano de Dinamic como *Yength* (1984) o *Arquímedes XII*. También se desarrollaban títulos totalmente en castellano, algunas veces versiones de juegos anglosajones y otras, pro-

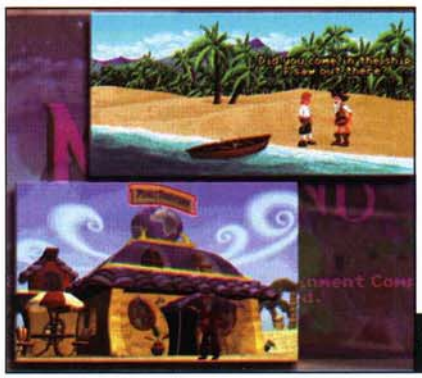
1

Mystery House (Arriba) Primera aventura gráfica. King´s Quest (abajo).



2

Otros éxitos de Sierra On Line: La serie Space Quest (arriba) y Gabriel Knight (abajo).

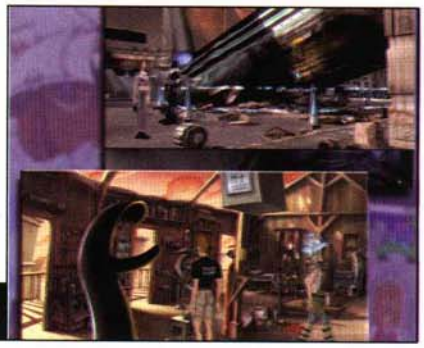


3

La serie de aventuras gráficas más famosa del mundo: Monkey Island.

4

El tesoro de la isla Alcachofa (arriba) y Sam & Max (abajo), dos ejemplos de perfecta animación en 2D.



5

Dos ejemplos de aventuras modernas con técnicas en 3D. (Arriba) The Longest Journey y Runaway: a road adventure (abajo).

LA BIOGRAFÍA...

ROBERTA WILLIAMS

Creadora de las **aventuras gráficas**

Empezó en el mundo de las aventuras jugando con *Colosal Cave Adventure* en papel. Le gusta comer bien y ver películas. Pero además de esto le gusta a hacer aventuras gráficas. Junto a su marido Ken fundaron Sierra On-Line después de escribir la primera aventura gráfica de la historia: *Mystery House* en 1980. Sus historias se caracterizan por estar siempre orientadas a todo el público y su mayor logro ha sido la serie *King´s Quest* con más de ocho entregas. Sus esfuerzos actuales se centran en las aventuras gráficas en internet.

Roberta Williams

ductos propios. Y todo de la mano de la compañía Aventuras AD, creadores de títulos sólidos como *La aventura original* (1988), *La diosa de Cozumel* (1990) o *Chichén Itzá* (1992)

Sierra seguía siendo la principal productora de aventuras gráficas para ordenador con títulos impecables considerados clásicos del género como la serie *Space Quest o Gabriel Knight*. El primer título de *Space Quest* se basaba todavía en un sistema híbrido pero con la posibilidad de mover el personaje con los cursores del teclado. Las órdenes se seguían dando a través de comandos. Pero cada vez se usaban verbos (cada vez con más posibilidades) para interactuar con los demás elementos del juego.

Otra compañía entró en el mundo de las aventuras gráficas para proporcionar títulos ya clásicos para los PC como *Maniac Mansion*, nos referimos a Lucas Arts. Después de esta aventura, desarrolla el primero de una saga de títulos que marcarían historia: *The Secret of Monkey Island* (1990). Poseía gráficos de 16 colores (aunque apareció posteriormente con 256) con una animación destacable en todos sus aspectos. Era el título que más gráficos utilizaba y sin duda, con el guión más audaz y divertido del género. A partir de los noventa el sistema de interacción a través de comandos verbales se sustituyó por el uso exclusivo del ratón. La riqueza de posibilidades se reducía a acciones determinadas impuestas por los gráficos de la pantalla. Las aventuras gráficas empezaron a coger un rumbo de éxito imparable. Los guiones se hacían más densos y complejos gracias al uso de sistemas de almacenamiento masivo como el CD. Las empresas pioneras optaron por completas bandas sonoras y gráficos espectaculares. Pero todo evoluciona y la tercera dimensión llamaba a las puertas del género cada vez más.

Pero el clásico 2D seguía produciendo títulos con animaciones de calidad como: *El tesoro de la Isla Alcachofa* de Alcachofa Soft o *Sam & Max* y *Day of the Tentacle* (1993, Lucas Arts). Incluso se desarrollaron títulos cuyos protagonistas eran actores reales como la aventura *Morpheus* (2000).

EL PASO A LAS 3D

Definitivamente las mejoras técnicas y cómo no, las exigencias del mercado, llevaron a la clásica representación en 2D de las aventuras gráficas a los entornos en 3D. Sin embargo, este paso no fue del todo inmediato y ni siquiera absoluto en un principio, ya que se opta en muchas ocasiones por el uso híbrido de las dos representaciones mezclando fondos prerenderizados con personajes en 3D y viceversa. Aun así, hoy día prevalecen las 3D e incluso clásicos como *Simon* o *Mokey Island* se desarrollan completamente en 3D. A partir del 2000 se desarrollan menos aventuras gráficas pero la mayoría son producciones enormes con guiones extensísimos y complejos y con gráficos muy detallados en 3D en las que incluso se podía rotar la cámara 360 grados como en *Myst III: Exile*. Encontramos títulos como *Schizm: Misterious Journey* (LK, Avalon, 2002) o *The Longest Journey* (Fun Com, 2000).

Pocas compañías se aventuraron a ir mas allá de lo impuesto gráficamente, a excepción de Péndulo que desarrolló *Runaway: A Road Adventure* (Dinamic Multimedia, 2000). Un título en el que se utilizan técnicas 3D pero dotando a los gráficos de un aspecto *cartoon* (dibujos animados) y con grandes resoluciones de pantalla.

En el próximo número...

... entraremos en el mundo de la simulación con el género de simuladores de vuelo.

Cuestionario Videojuegos 15

Preguntas

1. Escribe un bucle en el que salimos pulsando la tecla "Q".
2. ¿Qué comando debemos utilizar para saber en qué dirección se ha movido la rueda que algunos tipos de ratones llevan como tercer botón?
3. ¿Cómo podemos situar un grupo de objetos aleatoriamente en una zona determinada del terreno?
4. Cuando leemos el fichero que contiene los datos de una zona editada, ¿cómo podemos controlar cuándo no queda ningún dato más en el fichero?
5. ¿Con qué herramientas podemos crear un cubo y situarlo frente a la cámara?
6. ¿Dónde podemos asignar y modificar una textura y cómo se accede?
7. En Anvil Studio, nos situamos en el modo *Piano Roll* de la sección *Compose*. ¿Desde dónde podemos seleccionar un grupo de notas?
8. ¿Cuáles son los procedimientos para grabar una pista del CD en una pista de audio de Anvil Studio?
9. ¿Qué dos sistemas podemos utilizar para grabar y leer datos?
10. ¿Cómo podemos escribir y leer directamente en memoria?

Respuestas al cuestionario 14

▷ **1.**
```
AmbientLight 100,20,50
Direccional=CreateLight()
Puntual=CreateLight(2)
Foco=CreateLight(3)
```

▷ **2.** Mediante la instrucción
```
LightConeAngles Luz, Angulo interior#, Angulo exterior#
```

▷ **3.** Es una zona del entorno 3D donde se crean, evolucionan y destruyen partículas generalmente formadas por sprites.

▷ **4.**
```
Type emisor
Field X_Emisor#,Y_Emisor#,Z_Emisor#        ; Posición del emisor
Field RangoX#,RangoY#,RangoZ#              ; Tamaño del emisor
Field Entidad                              ; Entidad Sprite que forma la partícula
Field Vida                                 ; Vida de la partícula una vez emitida
Field Angulo_Rotacion#                     ; Angulo de rotación de cada partícula
Field Xvel#,Yvel#,Zvel#                    ;  Velocidad inicial de las partículas
End Type
```

▷ **5.** Se puede obtener con el programa LithUnwrap seleccionando todo el modelo y eligiendo la opción *Tools / UVMapping*.

▷ **6.** Aplicándola a una primitiva como un cubo, acercando una de sus caras a la cámara y guardando la imagen del render resultante.

▷ **7.** Nos situamos en *Compose* en el modo *Staff*. Nos situamos al comienzo del compás a partir del cual queremos insertar los compases. Pulsamos dos veces sobre el botón de insertar y escribimos "5" en la casilla de petición.

▷ **8.** Dejando pulsada la tecla "Ctrl" y sin dejar de hacer clic sobre la nota, desplazar el ratón.

▷ **9.**
```
Function ordenar_tabla()
  For n= 0 To  19
    For m= 0 To 19
      If Numero(m) < Numero(n)
        Aux_Numero= Numero (m)
        Numero (m)= Numero (n)
        Numero (n)=Aux_ Numero
      EndIf
    Next
  Next
End Function
```

▷ **10.**
```
Function Insertar_en_Tabla(numero)
  Aux_Numero=Numero
  For n=19 To 0 Step -1
    If Aux_Numero>Numero(n)
      Numero(n)=Aux_Numero
      Return
    EndIf
  Next
End Function
```

Contenido CD-ROM 15

▶ AUDIO

■ Tracktion 1.1.0

Controla proyectos musicales de principio a fin valiéndote de esta aplicación.

■ 1st Sound Recorder

Puede grabar en formato WAV o MP3 cualquier sonido que se produzca en tu ordenador, de forma automática y con alta calidad.

■ Advanced Sound Recorder 3.1

Esta aplicación complementa la anterior; es muy sencilla de usar y te permitirá capturar sonido fácil y rápidamente.

■ All Recorder 1.6.1

Graba, edita y modifica cualquier sonido capturado desde micrófono u otros programas en múltiples formatos.

■ Music MasterWorks 3.75

Potente programa para edición y secuenciamiento de archivos MIDI.

■ SpinStation Jukebox

Conviértete en un auténtico DJ usando todas las opciones de esta completa herramienta.

▶ DISEÑO 2D

■ Photomatte Lab 1.0

Completo programa para manipular imágenes y cambiarles las texturas.

■ Art Directors Toolkit 3

Aplicación directa cargada de prestaciones que te permite hacer un montón de cosas con tus imágenes.

■ DigiPicNamer 1.0

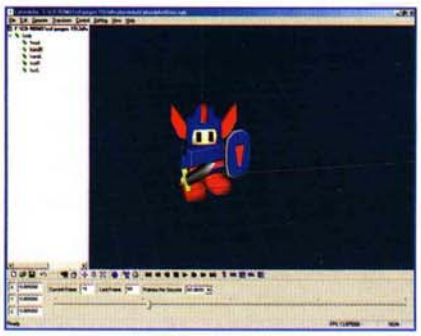
Si vas a manejar muchas imágenes al mismo tiempo, y quieres renombrarlas, podrás hacerlo de golpe con esta utilidad.

■ Photopleasure

Crea álbumes de imágenes o impórtalos a través de este simpático programa.

▶ DISEÑO 3D

■ Cyberdelia 2.5



Con esta aplicación modelarás en tres dimensiones de un modo muy sencillo.

■ 360 Professional Suite

Crea completas panorámicas en tres dimensiones fácil y rápidamente.

■ Bryce 5

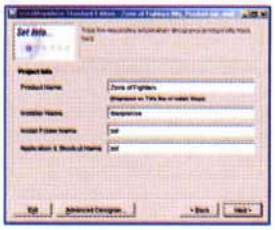
Nueva oportunidad de conseguir este programa, que usamos activamente en el curso.

■ Extreme Morph 3D 1.1

Otro programa de modelado 3D, para realizar primitivas y texturizarlas.

▶ PROGRAMACIÓN

■ InstallAnywhere 5



Simplifica el proceso de instalar en cualquier plataforma, y se hace cargo de todos los detalles.

■ Astrum InstallWizard

Con esta aplicación podrás crear el instalador de tu juego en pocos minutos.

■ Fresh Breeze Intelligent Editor

Editor inteligente con el que podrás usar un montón de lenguajes distintos de programación.

■ PatchWise 3.25

Crea completos proyectos de principio a fin sin ninguna dificultad.

■ POV-Ray 3.5

Uno de los trazadores de rayos más célebres, que nos permite representar escenas imaginarias definidas mediante el uso de unos modelos.

▶ JUEGOS

■ Runaway

Demo del estupendo juego en tres dimensiones con aspecto de dibujos animados.

■ King's Quest 1

Demo del clásico juego de Roberta Williams con el que tan buenos momentos hemos pasado.

■ Leisure Suite Larry

Juego muy divertido que narra los escarceos eróticos de un simpático personaje.

■ Maniac Mansion

Otro de los clásicos dentro de las aventuras gráficas que todos recordamos.

■ Day of the tentacle

Aventura gráfica muy adictiva y divertida.

■ Zone of Fighters

Como siempre, nuestro juego tal y como va quedando.

▶ VÍDEO

■ FlyVCD 2.5.0

Crea vídeos fácilmente a partir de imágenes estáticas.

■ LEAD MCMP_MJPEG Codec

Conversor de formatos de vídeo muy sencillo de usar.

■ MovieWow

Con esta divertida aplicación podrás crear spots comerciales de tu juego.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.

Inteligencia artificial (I) Plantas y animales

Empezamos con esta entrega el desarrollo de una de las partes del juego más interesante, sutil y, sobre todo, fundamental: la inteligencia artificial.

Básicamente, el comportamiento de los habitantes de un juego parte de una serie de premisas establecidas de antemano y que luego evolucionan durante el desarrollo de la partida. Esta evolución determina un mayor o menor grado de inteligencia y depende de la cantidad de sentencias lógicas implementadas en las rutinas de control del personaje. En "Zone of Fighters" utilizaremos unos patrones fijos para programar el comportamiento de los animales y plantas que habitan en el terreno de juego, los cuales veremos en esta entrega.

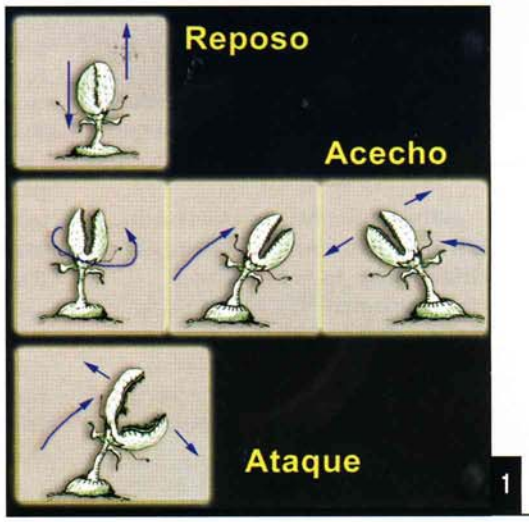
DANDO VIDA A LOS ANIMALES Y PLANTAS

Antes de estudiar el comportamiento de los seres vivos debemos inicializarlos en el juego.

En el módulo "funcpantaudio.bb" situamos la función "Crear_modelos()", la cual carga e inicializa cada una de las plantas y animales del juego, así como a la propia bionave de combate y el resto de elementos del decorado. Por motivos didácticos, vamos a trabajar a partir de ahora con modelos en formato .B3D, ya que es un formato más sencillo de utilizar para el manejo de modelos con animaciones. Sin embargo, es fácil cambiar el código para utilizar modelos en formato .MD2 o .X. Para cada modelo, vamos a extraer cada una de las animaciones que forman sus movimientos. Los utilizaremos posteriormente en la función que controla su comportamiento (Ver Código 1).

Después de cargar los modelos debemos situarlos en el terreno de juego.

Como vimos en el número anterior, en la función "Crear_Decorado()" incluimos las llamadas a las funciones que colocan a los animales y plantas sobre el terreno de juego. Estas funciones están situadas en el módulo "Enemigos.bb" y funcionan exactamente igual que las desarrolladas para los edificios y otros elementos del decorado.



Diseño de los diferentes estados que toma una planta carnívora.

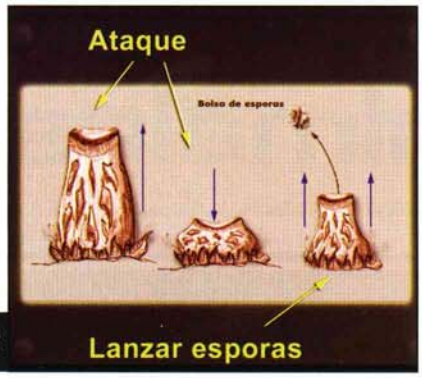
Código 1. Extraemos las animaciones de los modelos

```
mesh_dreeck=LoadAnimMesh("c:\zone of fighters\modelos\dreeck.b3d")
ExtractAnimSeq(mesh_dreeck,0,30)       ; reposo
ExtractAnimSeq(mesh_dreeck,31,90)      ; acechar
ExtractAnimSeq(mesh_dreeck,91,140)     ; ataque
; ---------------------------------------------------------------
mesh_luny=LoadAnimMesh("c:\zone of fighters\modelos\luny.b3d")
ExtractAnimSeq(mesh_luny,0,30) ;reposo y lanzar esporas
ExtractAnimSeq(mesh_luny,0,60) ;ataque
; ---------------------------------------------------------------
mesh_slunk=LoadAnimMesh("c:\zone of fighters\modelos\slunk.b3d")
ExtractAnimSeq(mesh_slunk,0,30)      ; salir de la tierra y entrar
ExtractAnimSeq(mesh_slunk,0,90)      ; ataque y a tierra
ExtractAnimSeq(mesh_slunk,80,90)     ; a tierra
```

COMPORTAMIENTO DE LAS PLANTAS CARNÍVORAS

La implementación del comportamiento de los Dreecks la realizamos en la función "Actualizar_dreecks()" situada en el módulo "Enemigos.bb". Esta función tendrá que ser llamada en cada bucle de juego, por lo tanto habrá que incluirla en la función "actualizar_juego()" del módulo "Fjuego.bb":

```
Function actualizar_juego()
    ...
    actualizar_dreecks()
    actualizar_slunks()
    actualizar_lunys()
    ...
End Function
```



2

Diseño de los diferentes estados que toma un Luny.

Como se había diseñado, las plantas carnívoras tendrán diferentes movimientos dependiendo de su estado. Estos movimientos son los que determinan, en cierta manera, su comportamiento, y son tres: reposo, acecho y ataque.

Cada uno de estos estados se activará según ciertas reglas que programamos en la función "Actualizar_dreecks()". Esta función consta de un bucle principal que recorre cada una de las plantas creadas y que forman parte de la estructura "tipo_dreeck":

```
Function actualizar_dreecks()
    For dreeck.tipo_dreeck=
    Each tipo_dreeck
    ...
```

El control del comportamiento de la planta se basa en dos factores. Por un lado, un contador interno que determina el tiempo de un estado u otro y por otro, la proximidad de la bionave de combate. Aunque colocáramos un Dreeck en un terreno totalmente vacío debemos asignarle un comportamiento base. En él determinaremos qué movimiento debe tener. Según el diseño de esta criatura, posee la habilidad de girar y alargar el tallo para buscar y alimentarse de cualquier presa que circule en sus proximidades. Determinamos entonces una postura primaria que será de reposo y otra de acecho cuando busca alimento. Estas dos posturas se activan de forma cíclica al paso de un determinado tiempo que controlamos con el contador "tiempo_cambio" y que forma parte de la estructura de la entidad:

```
Type tipo_dreeck
  Field x_dreeck#,z_dreeck#, size#
  Field tiempo_cambio,
  tiempo_cambio_total,
  Field ataque%, distancia_comida#,
  sw_movimiento
  Field entidad_dreeck, vida_dreeck
End Type
```

Este contador es inicializado con un valor aleatorio entre 200 y 400 cuando creamos la planta en la función "Colocar_dreeck()". Cambiando este valor determinamos más o menos tiempo en cada estado:

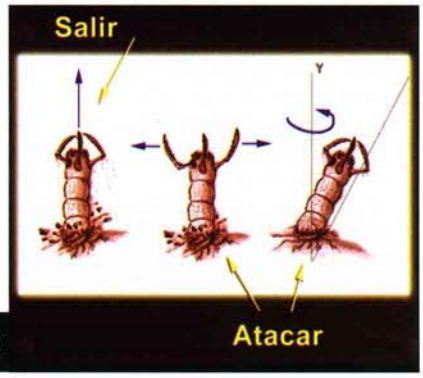
```
dreeck\tiempo_cambio=Rnd(200,400)
dreeck\tiempo_cambio_total=
dreeck\tiempo_cambio
```

De vuelta al control del comportamiento en la función "actualizar_dreecks()", este contador irá disminuyendo en una unidad cada vez que se llama a la función:

```
dreeck\tiempo_cambio=dreeck\
tiempo_cambio-1
```

Como esta variable perderá su valor inicial, necesitamos otra auxiliar que lo conserve, ya que lo necesitaremos para algunas comparaciones y que llamamos "tiempo_cambio_total".

Básicamente, lo que haremos es que el Dreeck cambie de postura (animación) cuando este contador llegue a un determinado valor. Necesitamos otras dos variables más para controlar la situación actual de la planta: un *switcher* que almacene el tipo de movimiento que está activo y que llamamos "sw_movimiento" (esta variable forma parte de la estructura de la entidad y toma dos valores: 1 ➡ planta en reposo y 2 ➡ planta en acecho) y "ataque", que indica si la planta está en ese momento en el estado de ataque (toma los valores 0 y 1) y que también forma parte de la estructura. Así, la planta estará en reposo cuando el contador "tiempo_cambio" llegué al cuarto de su valor inicial y



3

Diseño de los diferentes estados que toma un Slunk.

4

Utilización de "EntityDistance" para detectar la proximidad de la bionave a una planta carnívora.

```
Decrementar tiempo_cambio

Si distancia a bionave < 200 ATACA

Si tiempo_cambio <
   tiempo_cambio_total / 2  ACECHA

Si tiempo_cambio <
   tiempo_cambio_total / 4  REPOSO

Si tiempo_cambio = 0
   INICIALIZAR CICLO
```

5

Esquema del funcionamiento de la función que controla el comportamiento de un Dreeck.

mientras no esté atacando ("dreeck\ataque=0") (Ver Código 2).

Y cambia al estado de acecho cuando llegue a la mitad de su valor inicial (Ver Código 3).

Debemos tener en cuenta en cualquier momento de la vida del Dreeck la proximidad de la bionave de combate. Esto quiere decir que la planta cambiará al estado de ataque en el momento que divise comida. Para definir un rango de proximidad determinado a la bionave, utilizamos la instrucción "EntityDistance", la cual devuelve la distancia entre dos entidades:

```
dreeck\distancia_comida#=
EntityDistance (dreeck\
entidad_dreeck, bionave)
```

Por lo tanto, asignamos esta distancia a la variable de la entidad "dreeck\distancia_comida".

Luego aplicamos el ataque si la distancia es inferior a 200 puntos, siempre que la bionave no lleve activado el camuflaje ("camuflaje_bionave=0") y no esté atacando ya ("dreeck\ataque=0") (Ver Código 4).

Evidentemente, asignamos un 1 a la variable "ataque" para decirle a la función que la planta está en ese momento en estado de ataque. Como se puede observar, prácticamente lo que se hace es cambiar de animación. El resto lo hará el sistema de colisiones.

Para finalizar, hay que volver a inicializar todas las variables y el estado de la planta cuando el contador "tiempo_cambio" haya llegado a 0, volviendo a comenzar el ciclo (Ver Código 5).

No olvidemos que, al margen de estos controles de estado, es necesario controlar en esta función la vida de la planta y si ha recibido el impacto de cualquier disparo (Ver Código 6).

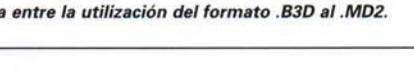
Observamos cómo la vida depende del tipo de disparo que haya impactado, determinado por la variable "tipo_armamento".

```
dreeck=LoadAnimMesh("dreeck.b3d")
ExtractAnimSeq(dreeck,0,30)
ExtractAnimSeq(dreeck,31,90)
ExtractAnimSeq(dreeck,91,140) ; 3
...
Animate dreeck,Modo,Vel,3,Trans
```

```
dreeck=LoadMD2("dreeck.md2")
AnimateMD2 dreeck,Modo,Vel,91,140
```

6

Diferencia entre la utilización del formato .B3D al .MD2.



NOTA

Incluir los contadores y demás variables como los *switchers* en la estructura "tipo.dreeck" permite independizar la actividad de cada planta colocada en el terreno de juego.

Código 2. Estado de reposo de la planta

```
;reposo
If (dreeck\tiempo_cambio<dreeck\tiempo_cambio_total/4 And dreeck\sw_movimiento=1) And dreeck\ataque=0
   Animate dreeck\entidad_dreeck,1,Rnd(.3,.5),1,30
   dreeck\tiempo_cambio_total=dreeck\tiempo_cambio
   dreeck\sw_movimiento=2
EndIf
```

Código 3. Estado de acecho de la planta

```
If (dreeck\tiempo_cambio<dreeck\tiempo_cambio_total/2 And dreeck\sw_movimiento=0) And dreeck\ataque=0
   Animate dreeck\entidad_dreeck,1,Rnd(.3,.5),2,30
   TurnEntity dreeck\entidad_dreeck,0,3,0
   dreeck\sw_movimiento=1
EndIf
```

Código 4. La planta ataca si no hay camuflaje

```
;ataca si no hay camuflaje
If dreeck\distancia_comida#<200 And dreeck\ataque=0 And camuflaje_bionave=0
    Animate dreeck\entidad_dreeck,1,Rnd(.3,.5),3,30
    dreeck\ataque=1
EndIf
```

Código 5. Vuelve a comenzar el ciclo

```
If dreeck\tiempo_cambio=0
        dreeck\tiempo_cambio=Rnd(200,400)
        dreeck\tiempo_cambio_total=dreeck\tiempo_cambio
        dreeck\sw_movimiento=0
        dreeck\ataque=0
        Animate dreeck\entidad_dreeck,1,Rnd(.3,.5),1,30
EndIf
```

Código 6. Controlamos la vida de la planta

```
If dreeck\vida_dreeck<1 Then
   puntos=puntos+200
   FreeEntity dreeck\entidad_dreeck
   Delete dreeck
   Return
EndIf
If EntityCollided(dreeck\entidad_dreeck,ENTIDAD_DISPARO)
   dreeck\vida_dreeck=dreeck\vida_dreeck-(tipo_armamento*3)
   puntos=puntos+20
EndIf
```

Código 7. En los Lunys, sustituimos tiempo_cambio por tiempo_ataque

```
If luny\tiempo_ataque>0
    luny\tiempo_ataque=luny\tiempo_ataque-1
    TurnEntity luny\entidad_luny,0,1,0
   ;ataque
    If luny\tiempo_ataque<luny\tiempo_ataque_total/2 And luny\
    sw_movimiento=0
       Animate luny\entidad_luny,1,.8,2,30
       luny\sw_movimiento=1
    EndIf
   ;lanzar esporas
    If luny\tiempo_ataque<luny\tiempo_ataque_total/4 And luny\
    sw_movimiento=1
       emisorparticulas3( ......................
       Animate luny\entidad_luny,1,.3,1,10
       luny\sw_movimiento=2
    EndIf
EndIf
```

Código 8. Movimiento de los Slunks

```
If slunk\tiempo_cambio>0
   slunk\tiempo_cambio=slunk\tiempo_cambio-1
  ;sale, ataca y entra en la tierra
   If slunk\tiempo_cambio<slunk\tiempo_cambio_total/2 And slunk\
   sw_movimiento=0
   Animate slunk\entidad_slunk,3,Rnd(.3,.5),2,30
   slunk\sw_movimiento=1
   EndIf
EndIf
```

COMPORTAMIENTO DE LOS DEMÁS SERES

El sistema utilizado para las plantas carnívoras es el que utilizamos para los Slunks y Lunys. La única diferencia son los tipos de estado de cada criatura. Como las variables de control de estado (contadores y switchers) pertenecen a la estructura de cada tipo de entidad, tendrán el mismo nombre y misión dentro de la función. Los Lunys no detectan la proximidad de la bionave, solo varían su estado según el control del contador "tiempo_cambio".

(Para los Lunys, la variable "tiempo_cambio" es sustituida por "tiempo_ataque") (Ver Código 7).

Los Slunks tampoco detectan la proximidad de la bionave, ya que su único estado en movimiento agrupa toda la activa en una sola animación. También viene determinado por el contador "tiempo_cambio" (Ver Código 8).

Sin embargo, añadir una detección de proximidad es fácil cambiando la sentencia condicional de este modo:

```
If (slunk\tiempo_cambio<slunk\
tiempo_cambio_total/2 And
slunk\sw_movimiento=0) Or
 EntityDistance (dreeck\
 entidad_slunk,bionave) < 200
```

Como hemos podido comprobar, implementar un comportamiento básico es realmente sencillo y todo depende de sentencias condicionales que detecten los cambios de la acción y, como siempre, de nuestra imaginación.

En el próximo número...

... seguiremos desarrollando la IA del juego aplicando el comportamiento de los OVNIS y los VOLADORES.

Fabricando el terreno de juego

En esta entrega vamos a aprender cómo realizar de varias formas el terreno para nuestro juego.

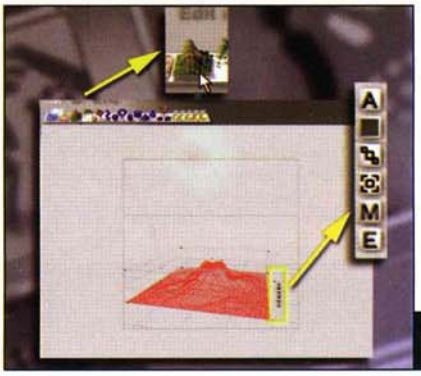
CONCEPTOS BÁSICOS

Básicamente, hay dos maneras de disponer de un terreno en Blitz3D: utilizando un *mesh* y mediante un mapa de alturas. La utilización de un modelo 3D (*mesh*) es la manera más natural de disponer de un terreno en nuestro juego. Este método tiene la ventaja de proporcionar un mayor control de todos los polígonos pero es más complicado y laborioso. Es muy utilizado para realizar, por ejemplo, circuitos para juegos de carreras de rally o simulaciones precisas de espacios abiertos. Con este sistema se puede crear un terreno a partir de un solo modelo o mediante la unión de varios; por ejemplo, podemos crear una cordillera uniendo diferentes modelos 3D de montañas. El segundo de los métodos (el mapa de alturas) es más sencillo de utilizar y consiste en representar alturas a partir de tonos de grises contenidos en un mapa de bits. Para crear el terreno en "Zone of Fighters" vamos a utilizar ambos métodos, los cuales desarrollaremos con la aplicación Bryce 5 y Paint Shop Pro. Sin embargo, aprovechando la cualidad de Blitz3D para generar un *mesh* a partir de un mapa de alturas, nos quedaremos al final con el segundo de ellos.

El programa Bryce es ideal para la realización de terrenos utilizando mallas a partir de un mapa de alturas. Sin embargo, solo permite exportar el *mesh* y las texturas que le apliquemos al terreno y no el mapa de alturas en sí. Pero nos ayuda a tener una referencia de cómo será nuestro terreno. Por otro lado, en Paint Shop Pro podemos dibujar nuestro mapa de alturas directamente e ir probando el resultado con Blitz3D o Bryce como veremos más adelante. Otra opción es ayudarnos de la imagen de las texturas que genera Bryce para dibujar en Paint Shop Pro por el método de capas superpuestas.

CREANDO NUESTRO TERRENO CON BRYCE. USANDO UN *MESH*

Ejecutemos la aplicación Bryce 5. Una vez dentro, lo primero que tenemos que hacer es crear un documento nuevo de 512 x 512 (*File/New Document* - "Ctrl" + "N") o modificar el existente por defecto (*File/Document Setup* - "Ctrl"+"Alt"+"N"). En la ventana *Document Setup* no olvidemos deseleccionar *Constrain Proportions*. La siguiente operación será crear la malla del terreno pulsando en el icono del menú *Create*. Bryce genera automáticamente un nuevo terreno con forma diferente cada vez que se pulsa en este icono. Entonces podemos observar cómo aparece una malla con la forma de una montaña en la vista. Para obtener una forma determinada de la malla debemos editarla. Para ello, pulsamos sobre la letra "E" que aparece en los iconos de



1

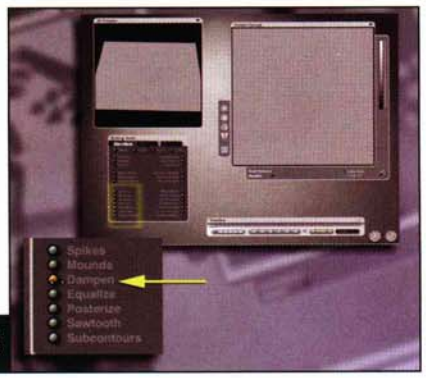
Procedimientos para crear un terreno y entrar en su editor.

2

En la ventana 3D Preview podemos ver la representación 3D del mapa de alturas desde todas direcciones.

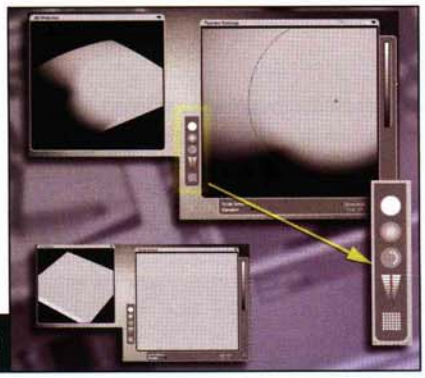
3

Descripción de las herramientas del pincel.



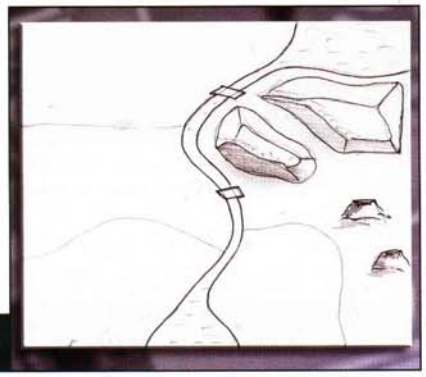
4

En la pestaña Elevation de la ventana Editing Tools encontramos todos los efectos que podemos aplicar al mapa de altura.



5

Antes de dibujar las diferentes alturas es necesario dibujar un terreno con una altura media uniforme.



6

Diseño original del terreno que tenemos que crear.

RECORDATORIO

Es importante recordar que, en una imagen para un mapa de alturas, las tonalidades negras representan las partes más bajas del terreno y las blancas las más altas.

opciones situados en la parte inferior derecha del terreno creado y entraremos en el editor de terrenos de Bryce. En él podemos distinguir varias ventanas (Ver Fig. 1).

VENTANA *3D PREVIEW*

Esta ventana nos muestra un previo de cómo será el terreno que estamos editando. Navegar por la vista es muy sencillo. Si queremos rotar el terreno, hacemos clic sobre la ventana y sin dejar de hacerlo movemos el ratón. Para acercarnos o alejarnos, movemos el ratón hacia delante o atrás, manteniendo pulsada la tecla "ALT". Para aumentar o disminuir la altura del terreno, movemos el ratón hacia delante o atrás mientras mantenemos pulsada la tecla "CTRL". Y para activar el "Auto Rotate", es decir, que el terreno gire sobre su eje Y, hacemos clic mientras mantenemos pulsada cualquier tecla "SHIFT". Pulsando sobre la flechita negra situada en la esquina superior derecha de la ventana entraremos en una serie de opciones en la que podemos cambiar la resolución de la ventana (*Set Preview Size*), activar el *Auto Rotate*, hacer un renderizado del previo, etc. (Ver Fig. 2).

VENTANA *TERRAIN CANVAS*

En esta ventana es donde vamos a dibujar nuestro mapa de alturas. Podemos observar cómo el terreno se representa como un dibujo de manchas blancas y negras. Realmente, es una zona de dibujo en la que podemos pintar con el ratón. A la izquierda de la ventana tenemos las opciones de pintado, las cuales describimos en la figura 3.

Pulsando en la flechita de color negro situada en la esquina superior derecha de la ventana entramos en el menú de opciones, en donde podemos modificar su tamaño. Al lado de esta flechita encontramos otro icono y al pulsarlo mostraremos algunas opciones más, entre ellas las del comportamiento del pincel.

VENTANA *EDITING TOOLS*

En esta ventana encontramos todos los efectos que le podemos aplicar al mapa de alturas, así como la asignación de imágenes o filtros. En nuestro terreno, solo vamos a utilizar efectos de elevación ("Elevation").

DIBUJANDO NUESTRO TERRENO

Vamos a comenzar realizando el mapa de alturas de nuestro terreno. Pero antes vamos a cambiar la resolución de la ventana de dibujo. Para ello, pulsamos en el icono de resolución y elegimos "512 – ultra-fine". El siguiente paso será rellenar toda la imagen con una altura media para, posteriormente, ir creando las partes altas y bajas del terreno. Para realizar esta operación podemos utilizar varias maneras. Una de ellas es aplicar un efecto *Dampen* (situado en la pestaña *Elevation* de la ventana *Editing Tools*) y la otra borrando el terreno que tenemos por defecto pulsando en el botón "New" situado en esta misma ventana. Para aplicar el efecto *Dampen*, nos situamos sobre la esfera verde situada junto al nombre "Dampen" y sin dejar de hacer clic desplazamos el ratón hacia el lado izquierdo hasta que veamos en la ventana *Terrain Canvas* que toda la imagen se vuelve de un tono uniforme gris. A partir de ahí, dibujamos las diferentes alturas (Ver Fig. 4).

Si por el contrario decidimos utilizar una imagen nueva, debemos pintar toda la ventana de dibujo con una altura uniforme. Para ello, colocamos el punto rojo de altura del icono en la mitad

de la escala. Seguidamente, seleccionamos la anchura máxima del pincel en el icono de tamaño, la máxima dureza (segundo icono) y un flujo de tinta máximo (tercer icono) (Ver Fig. 5).

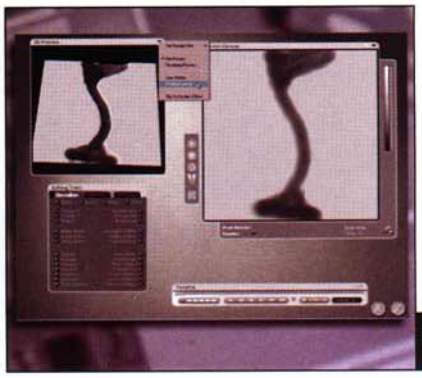
Ya estamos preparados para definir todas las alturas del terreno de combate.

En primer lugar, vamos a dibujar el río que atraviesa el terreno según el diseño original (Ver Fig. 6) para dividir las dos partes del terreno. Elegimos un tamaño pequeño de pincel, la mínima dureza y un flujo medio de tinta. Y para finalizar, seleccionamos la altura mínima. Dibujamos con cuidado hasta obtener el resultado que podemos ver en la figura 7.

El siguiente paso será rodear el terreno de un muro. Así que tenemos que elegir la altura máxima (color blanco). Básicamente, ya tenemos la forma principal del terreno terminada. Nos queda dibujar los soportes para los puentes, las rampas de acceso desde el río y algunas montañas, entre ellas los volcanes. Recordaremos que la suavidad de bordes en el dibujo determinará unos cambios de altura redondeados. Para que el terreno no sea uniforme, podemos variar las alturas colocando algunas pinceladas de diferentes tonos. También sería interesante aplicar un efecto de erosión con la opción *Eroded* en la ventana *Editing Tools*. Hay que tener en cuenta que la escala del dibujo es muy pequeña con relación a la que necesitaremos en el juego. Por lo tanto, cualquier fisura o cambio de altura en el terreno se verá multiplicada en gran medida cuando estemos visualizándolo con Blitz3D. Básicamente, el trabajo que nos queda es conformar las diferentes partes del terreno, siempre siguiendo el diseño original. Por ejemplo, para el desfiladero, seleccionamos la altura máxima y pintamos. Luego, aplicamos un poco de erosión en los bordes cambiando el comportamiento del pincel en *Brush Behavior*. Para ello, pulsamos en la flechita negra situada junto a *Elevation*. Aparece un menú con los distintos comportamientos que puede adquirir el pincel y elegimos *Erosion*. La cantidad de erosión viene determinada por el tamaño del pincel. Para realizar los volcanes, solo tenemos que colocar un punto blanco (máxima altura) y otro más pequeño de color negro (mínima altura) en el centro para representar el cráter (no olvidemos volver a colocar el comportamiento del pincel en la opción *Elevation*). Antes de terminar no olvidemos dibujar los soportes para los puentes (Ver Fig. 8).

EXPORTANDO EL TERRENO

Una vez terminado de dibujar el terreno ya podemos exportar la malla que lo representa. Pero antes, es necesario aplicarle una textura. Volvemos a la vista principal pulsando en el icono de aceptar y entramos en el laboratorio de materiales. Por ejemplo, elegimos el material *Mid-Winter* de la librería *Planes & Terrains*. Una vez aplicado el material, volvemos a la zona de edición. Elegimos la opción *Export* situada en el menú (flechita negra) de la ventana *Terrain Canvas*. A continuación, el programa nos pedirá el nombre del archivo que contendrá el *mesh* del terreno. Seleccionamos el formato .3DS (versión 5) y un nombre. Al pulsar "Guardar" aparecerá la sección *Export Terrain*. Aquí, vemos una ventana en la que se muestra una representación 3D del terreno, que podemos rotar y mover como viene siendo habitual para todas las ventanas de previo. Vamos a modificar el número de polígonos del terreno a



7

Aspecto que debe tomar el río que atraviesa la zona de combate.

8

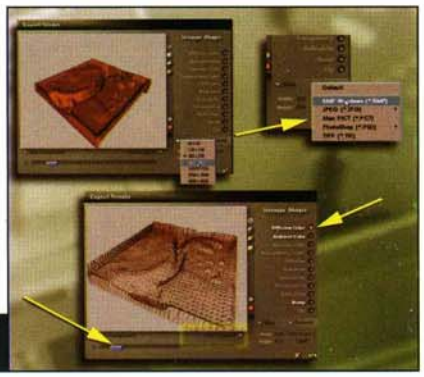
Aspecto definitivo del mapa de alturas.

NOTA

En Bryce 5 todas las ventanas del modo edición de terrenos se pueden desplazar como las propias de Windows.

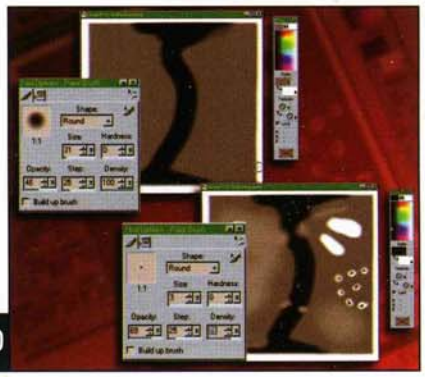
NOTA

Para visualizar el resultado en tiempo real en la ventana *3D preview* mientras estamos pintando, debemos activar la opción *Realtime Linking* del menú de *opciones* de dicha ventana. También, en esta ventana podemos apreciar un renderizado del terreno activando la opción *Rendered Preview*.



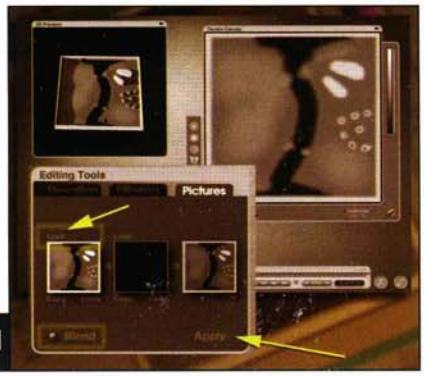
9

Bryce nos permite exportar el terreno como un mesh y también las distintas propiedades del material.



10

También es posible crear con Paint Shop Pro el mapa de alturas utilizando el pincel.



11

Podemos utilizar Bryce para visualizar un mapa de alturas dibujado con Paint Shop Pro.

3200 desplazando la barra deslizadora azul hacia la izquierda. A la derecha de la ventana encontramos el apartado *Image Maps*, el cual muestra todas las cualidades del material que se exportará en imágenes; es decir, si tenemos activado *Diffusion Color* y *Bump* obtendremos dos imágenes, una para cada propiedad. En *Size* elegimos el tamaño de las imágenes y en *Forma* el formato. Por defecto tenemos un tamaño de 256 x 256, así que cambiémoslo a 512 x 512. y elijamos el formato .bmp. Como comentábamos, estas imágenes se pueden utilizar como referencia en Paint Shop Pro para desarrollar el mapa de alturas. Como solo necesitaremos la capa *Diffusion Color* para obtener esta referencia, deseleccionamos el resto de las opciones del material (Ver Fig. 9).

CREANDO EL MAPA DE ALTURAS CON PAINT SHOP PRO

Este método es más artesanal ya que tenemos que dibujar las diferentes alturas utilizando tonos de grises. Para empezar, creamos un documento nuevo de 512 x 512. Seguidamente, seleccionamos un tono de color gris y rellenamos la pantalla con la herramienta de relleno. Prácticamente, el mapa se dibujará de la misma manera que en Bryce, eligiendo tonos grises claros para las elevaciones y tonos oscuros para las hondonadas. Es importante jugar con los parámetros del pincel para obtener trazos suaves y así evitar brusquedad en los cambios de tonos. Como siempre, lo primero que vamos a dibujar es el río en el centro y los muros que delimitan el terreno. A continuación, los soportes para los puentes, el desfiladero y los volcanes. Terminaremos, haciendo menos uniforme el terreno con algunos cambios de tono para simular montículos y hondonadas (Ver Fig. 10).

Para comprobar el resultado podemos utilizar el visualizador de terrenos que desarrollamos en el número 7 del curso, o bien utilizar Bryce.

COMPROBANDO EL RESULTADO EN BRYCE

Grabemos el mapa de alturas dibujado en Paint Shop Pro en formato .bmp y ejecutemos Bryce. Dentro del programa, asignamos un tamaño al documento de 512 x 512. Seguidamente, creamos un terreno y entramos a editarlo. En la zona de edición elegimos la opción *New* en la pestaña *Elevation* de la ventana *Editting Tools*. Luego pulsamos en la pestaña *Pictures*. Aparecerán tres ventanitas separadas por los signos "+" y "=". Esto representa la posibilidad de utilizar cualquier imagen o varias mezcladas para crear un terreno. Hacemos clic en *Load* de la primera ventana y cargamos el mapa de alturas que dibujamos con Paint Shop Pro. A continuación, hacemos clic sobre la segunda ventana hasta que la imagen desaparezca. Seguidamente, desplazamos el ratón hacia la izquierda sobre *Blend* (mezcla), sin dejar de hacer clic, para modificar la mezcla entre las dos ventanas. El resultado (tercera ventana) tiene que ser igual a la primera ventana. Si pulsamos sobre la ventana del resultado (3ª ventana) se mostrará en *3D Preview* una representación del terreno. Al pulsar en *Apply* se usará nuestra imagen como terreno actual (Ver Fig. 11).

En el próximo número...

...desarrollaremos las texturas que cubrirán el terreno.

Nuestro primer tema musical con Anvil Studio (V)

Siguiendo con nuestro aprendizaje de Anvil Studio, continuaremos viendo las posibilidades del manejo de pistas de audio en nuestras canciones.

En el número anterior aprendimos la forma de grabar audio externo desde un micrófono directamente en una pista. Veremos a continuación cómo trabajar con una pista de audio y cómo obtener y editar ficheros de audio.

CARGANDO Y EDITANDO UN FICHERO DE AUDIO

Podemos importar directamente un fichero de audio en formato .WAV a una pista del secuenciador. La versión estándar de Anvil Studio solo puede mezclar una pista de audio a la vez, para obtener más pistas es necesario adquirir el accesorio "MultiAudio". Sin embargo, es suficiente para nuestros propósitos. Cambiamos la pista por defecto de *Instrument* a *Audio* en el campo *Type* y elegimos *Stereo* en el campo *Channel*.

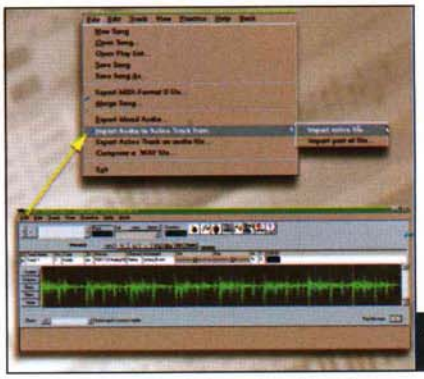
Activamos la pista y ya estamos preparados para recibir un fichero externo de audio. Seleccionamos la opción *Import Audio to Active Track from... / Import entire file* del menú *File*. Si no hemos salvado la canción actual nos pedirá un nombre de canción para guardarla y luego podremos elegir el fichero de audio. Importamos "loop.wav" contenido en "Extras" del CD (Ver Fig. 1).

Veremos más adelante cómo podemos utilizar ficheros de audio como sonidos para nuestras pistas de ritmos. Una vez cargado, podemos ver y editar la forma de onda pulsando en el botón "Compose". Para reproducir el audio pulsamos en el botón junto a "Play this track". Podemos también reproducir solo una parte del audio seleccionándola con el ratón (Ver Fig. 2).

A la izquierda de la ventana que muestra la forma de onda encontramos 5 botones: "Louder", "Volume", "Pitch", "Filter" y "Mixer". El primero de ellos ("Louder") sirve para doblar el volumen actual de la muestra (hay que tener en cuenta que este tipo de operaciones puede ocasionar distorsión en el audio). En "Volume" podemos modificar el volumen de la muestra (desde 0 hasta 3) de forma controlada a través de un deslizador. "Pitch" sirve para cambiar el tono de la onda. El botón "Filter" nos lleva a una librería de filtros para aplicar a la muestra. Entre ellos encontramos el normalizado, que como sabemos, es muy útil para subir al máximo el volumen de la muestra sin distorsionar. Con el último botón ("Mixer") salimos de *Compose* (Ver Fig. 3).

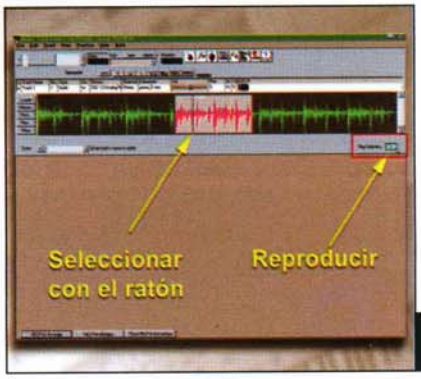
El deslizador "Zoom" que hallamos en la parte inferior nos ayuda a aumentar la muestra en el tiempo y el deslizador vertical situado justo al final de la ventana de visualizado de onda, sirve para aumentar la muestra en la gráfica del volumen. Otro efecto que podemos aplicar es dar la vuelta a la muestra o parte de ella con la opción *Reverse* del menú *Edit*. Par poder aplicarlo es necesario seleccionar parte de la muestra con el ratón o seleccionándola por completo con *Edit/Select All* ("Ctrl" + "A").

También es posible realizar operaciones de edición como



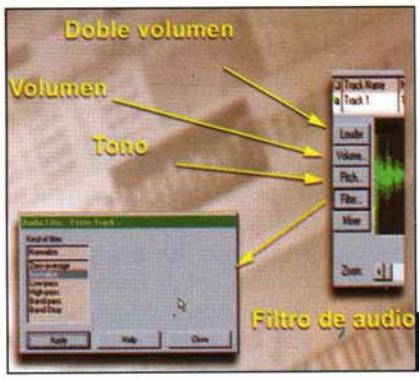
1

Podemos importar un fichero de audio en formato .WAV directamente a la pista.



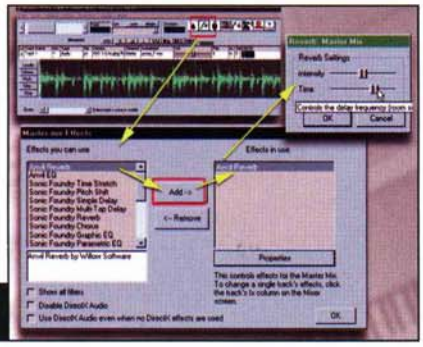
2

AL pulsar en el botón "Compose" entramos en la ventana de edición de la muestra.



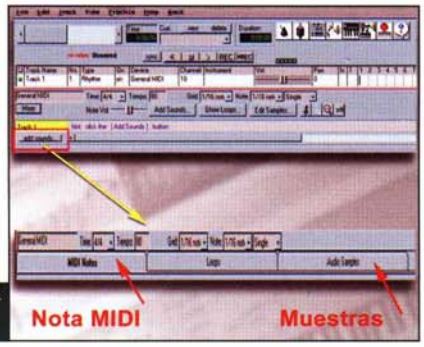
3

Significado de los diferentes botones de opciones para modificar la muestra de audio.



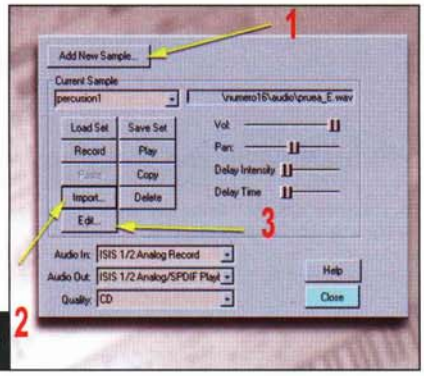
4

Además de los efectos propios de Anvil Studio, podemos añadir todos los efectos compatibles con DirectX instalados en el sistema.



5

Las pistas de ritmos son ideales para secuenciar partes con muestras y notas MIDI.



6

Pasos para añadir un nuevo sonido de muestra.

NOTA

Al cambiar la pista activa en pista de audio, podemos observar cómo en los iconos situados en la parte superior derecha aparece un icono nuevo, el cual sirve para acceder a los efectos de audio.

cortar, borrar, pegar o insertar con partes de la muestra. Para ello, solo tenemos que seleccionarla y elegir las opciones. Las operaciones de edición funcionan exactamente igual que en otros tipos de modos y pistas.

APLICANDO EFECTOS A LA PISTA DE AUDIO

Es posible aplicar otros tipos de efectos a la muestra completa o parte de ella. Solo tenemos que pulsar en el icono y entraremos en una ventana flotante con las librerías disponibles. Anvil Studio cargará automáticamente todos los efectos compatibles con DirectX que tengamos instalados en nuestro sistema además, claro está, de los efectos propios del programa. Para aplicar el efecto, debemos seleccionarlo de la lista *Effects you can use* y pulsar en el botón "Add —>" para pasarlo a la lista *Effects in use*. Para modificar los parámetros de los efectos, solo tenemos que pulsar en el botón "Properties" y se abrirá otra ventana flotante con sus cualidades (Ver Fig. 4).

CREANDO PISTAS DE RITMOS

El último tipo de pistas que podemos crear en Anvil Studio son las de ritmos. Estas pistas se pueden considerar como una mezcla entre "Instrument" y "Audio" ya que permiten mezclar notas MIDI y muestras. Creamos una canción nueva. En la pista por defecto cambiamos el tipo (Type) a pista de ritmo (Rhythm). Para editarla, pulsamos en el botón "Compose". Para poder añadir notas a la pista es necesario seleccionar antes un sonido. Para ello, pulsamos en el botón "add sounds..." situado en la parte inferior. Aparecerá la lista de notas MIDI en la pestaña "MIDI notes". Pero también podemos seleccionar loops o muestras, en las pestañas "Loops" y "Audio Samples" respectivamente (Ver Fig. 5).

Vamos a elegir un sonido perteneciente a una muestra. Pulsamos sobre la pestaña *Audio Samples* y después sobre *Edit Samples*. Aparece una nueva ventana en donde obtendremos la muestra importándola de disco o grabando directamente. En nuestro caso, vamos a cargar el sonido "HandClap.wav". Antes de poder importar debemos asignar mediante un nombre (por ejemplo "percusion1") la nueva muestra al programa pulsando sobre el botón "Add New Sample". Después de esto ya podemos cargar la muestra pulsando sobre el botón "Import...". Una vez cargada, podemos editarla (botón "Edit...") o reproducirla (botón "Play") antes de cerrar o también modificar directamente su volumen, panorámico, etc. mediante los deslizadores Vol, Pan, etc. (Ver Fig. 6).

Al pulsar "Close" volvemos a la pantalla *Audio Samples* en donde podemos apreciar que nuestra nueva muestra ha inaugurado la lista. Al volver a la cuadrícula vemos cómo debajo de "Track1" (en amarillo) aparece un botón con el nombre de la nueva muestra. Si pulsamos sobre él sonará. Ya solo tenemos que añadir sobre la cuadrícula las notas, haciendo clic con el ratón. Podemos sumar a la lista cuantos sonidos de muestras queramos, además, claro está, de sonidos MIDI.

Este sistema para añadir sonidos de muestras es también viable en pistas de tipo "Instrument"; la única diferencia es la forma de edición de este tipo de pistas, pero para todo lo demás es exactamente igual.

En el próximo número...

... terminaremos nuestro aprendizaje de Anvil Studio viendo el funcionamiento de la opción "Performer" y de cómo realizar Loops.

Manejo de ficheros y los bancos de memoria

Después de tratar en el número anterior la gestión del teclado, ratón y dispositivos de juego, en esta ocasión aprenderemos a utilizar los ficheros y todos los procesos derivados de su uso.

Debemos distinguir entre la gestión del fichero en sí mismo y el manejo de los datos que contiene. Por lo tanto, aprenderemos a crear, borrar, buscar, etc. ficheros en el disco, así como a trabajar con los directorios. Y por otro lado, veremos la manera de grabar o leer datos de un fichero.

Como sabemos, el uso de ficheros es fundamental en cualquier aplicación. Siempre es interesante poder grabar datos de utilización interna de un programa en un soporte físico para uso posterior o para intercambio con otras aplicaciones. Desde un sencillo archivo de configuración hasta una gran base de datos, todo es posible en Blitz3D. Además, se dispone de instrucciones para el acceso secuencial y aleatorio de los datos contenidos en un fichero, este último a través de punteros de localización realmente flexibles.

```
Dir= ReadDir("c:\")
Repeat
  F$=NextFile(Dir)
  Print F$
Until F$=""
```

1

Código para mostrar los ficheros contenidos en un directorio.

MANEJANDO DIRECTORIOS

Antes de trabajar con los ficheros individualmente, vamos a conocer cómo controlamos el uso de los directorios donde ubicaremos nuestros archivos. Empezaremos, conociendo cuál es el directorio actual seleccionado, en el que se situarán todas las operaciones con archivos. Para ello tenemos la instrucción "CurrentDir$()". Esta instrucción devuelve un *string* (lista de caracteres alfanuméricos) con el nombre del directorio actual en uso:

```
Dir_actual= CurrentDir$()
Print "El directorio actual
es: "+Dir_actual
```

Si queremos cambiarlo, disponemos de "ChangeDir":

```
ChangeDir "C:\juego\datos"
```

Aquí, cambiamos el directorio actual a "datos".

Crear o borrar un directorio es fácil con "CreateDir" y "DeleteDir" respectivamente:

```
Nuevo_Dir= "C:\juego\datos"
CreateDir Nuevo_Dir
DeleteDir Nuevo_Dir
```

Otra función interesante que podemos usar es "ReadDir", la cual permite leer el contenido de un directorio; es decir, todos los archivos y carpetas que contiene. Sin embargo, para sacar todo el provecho a esta instrucción debemos utilizarla en combinación con "NextFile$" o "FileType". No olvidemos que después de abrir un directorio y hacer operaciones con él es necesario cerrarlo con "CloseDir".

```
CurrentDir
ChangeDir
CreateDir
DeleteDir
ReadDir
CloseDir
```

2

Lista de comnados disponibles para el manejo de directorios.

OPERACIONES CON LOS ARCHIVOS DE UN DIRECTORIO

Como comentábamos, "ReadDir" solo lee el directorio, así que para leer cada archivo o carpeta del mismo debemos recorrerlo con la instrucción "NextFile$":

```
Dir=ReadDir("c:\")
Repeat
   F$=NextFile(Dir)
   Print F$
Until F$=""
```

"NextFile$" proporciona el nombre y la extensión del siguiente fichero del directorio leído por "ReadDir" pero no

NOTA

No podemos recorrer un directorio hacia atrás, solo hacia delante. Para poder trabajar con el directorio a nuestro antojo debemos introducir su contenido en una matriz. De esta forma, podemos, por ejemplo, ordenarlo o recorrerlo en cualquier sentido.

OpenFile
ReadFile
WriteFile
CloseFile
CopyFile
DeleteFile

3

Lista de comandos básicos para el manejo de ficheros.

sabe distinguir si se trata de un archivo o de una carpeta, para ello debemos utilizar la instrucción "FileType".

```
FileType (Nombre del fichero$)
```

Este comando devuelve 3 valores:

- **1.** El nombre del fichero existe.
- **0.** El nombre del fichero no existe.
- **2.** El nombre del fichero corresponde a un directorio.

```
Fichero$="C:\juego\config.dat"
If FileType(Fichero$)=0 Print
"El fichero de configuración
no existe"
```

NextFile
FilePos
SeekFile
FileType
FileSize
ExecFile

4

Lista de comandos especiales para el manejo de los datos de un fichero.

A menudo, necesitaremos saber el tamaño en bytes de un fichero, bien para copiarlo, leerlo o almacenarlo en un banco de memoria. Para disponer de esta información tenemos la instrucción "FileSize":

```
Fichero$="C:\juego\nivel.dat"
Tamaño = FileSize (Fichero$)
Nivel= CreateBank (Tamaño)
; Obtenemos el tamaño en bytes
```

Podemos también actuar directamente con los ficheros de un directorio determinado. Así, si queremos borrar un fichero, podemos hacerlo con "DeleteFile":

```
DeleteFile "C:\juego\datos.dat"
```

Es necesario asegurarse de que el fichero que vamos a borrar existe. Si no existe, el programa no dará error pero, evidentemente, no habrá ninguna consecuencia en el directorio.

Una opción muy útil: si queremos construir nuestro propio instalador, está la necesidad de copiar ficheros de una ubicación a otra. Para ello, Blitz3D nos proporciona la instrucción "CopyFile":

```
CopyFile ubicación origen$,
ubicación destino$
CopyFile "C:\pantalla.jpg"
"A:\pantalla.jpg"
```

Y por último, una instrucción que nos permitirá ejecutar un programa ejecutable externo, "ExecFile". Muy interesante, por ejemplo, para visualizar un fichero de texto desde Blitz3D, como la ayuda del juego, etc. (Ver "ejemplo1.bb"):

```
ExecFile ("C:\windows\
notepad.exe ayuda.txt")
```

MANEJANDO ARCHIVOS Y EDITANDO SU CONTENIDO

A continuación, vamos a aprender cómo crear y utilizar nues-

5

Esquema de la disposición de números enteros en un fichero.

tros propios ficheros para grabar o leer datos. Antes de cualquier operación de lectura o escritura en un fichero nuevo, debemos crearlo. Para ello, utilizamos la instrucción "WriteFile". En realidad, lo que este comando hace es abrir un fichero y lo prepara para que se puedan grabar datos en él. Si el fichero no existe, lo crea:

```
Fichero_Datos= WriteFile
("Datos.dat")
```

Si ya tenemos un fichero creado con datos y queremos leer de él, debemos utilizar "ReadFile" para abrirlo:

NOTA

Si abrimos un fichero con "WriteFile" y el fichero ya existe, inicializará el archivo borrando todos los datos aunque no hagamos ninguna operación en él.

RECORDATORIO

Vamos a recordar los valores que toman los diferentes tipos de datos en Blitz3D:

Byte	**0 - 255**
Short	**0 - 65535**
Int	**-2147483647 - 2147483647**
Float	**-3.4 x 10 ^ -38 - 3.4 x 10 ^ -38 .**

SeekFile Fichero, Desplazamiento

Desplazamiento =

Elemento x Tamaño del Elemento -

Tamaño del Elemento

6

Cálculo del desplazamiento para la búsqueda de registros en un fichero.

```
ReadFile (nombreFichero$).
```

Una vez que hayamos abierto un archivo y terminado las operaciones con él, es importante cerrarlo con la instrucción "CloseFile":

```
Datos = ReadFile ("Configuración.dat")
CloseFile Datos
```

Blitz3D nos permite leer o grabar un dato determinado de un fichero sin tener que realizar ninguna copia del fichero o trabajar con todos los registros secuencialmente. Disponemos para ello de "FilePos" y "SeekFile".

El primero de ellos, "FilePos", devuelve la posición actual que esta siendo editada (leída, escrita o modificada) dentro del fichero.

```
Fichero = WriteFile("datos.dat")
For n = 1 To 8
  Numero=Rand (1,100)
  WriteInt ( Fichero, Numero )
Next
CloseFile (Fichero)
; ----------------------------
Fichero = ReadFile("datos.dat" )
For n=1 To 8
  Numero = ReadInt(Fichero)
  Posicion = FilePos (Fichero) - 4
  Print "Numero: " + Numero +
  " Posición "+ Posicion
Next
CloseFile( Fichero )
```

En este ejemplo, creamos un fichero ("datos.dat") y grabamos en él 8 números enteros aleatorios. Seguidamente, volvemos a abrirlo para leer los 8 números. En las líneas:

```
Posicion = FilePos (Fichero) - 4
Print "Numero: " + Numero +
" Posición "+ Posicion
```

Imprimimos en pantalla el número contenido en el fichero y su posición. Debemos observar cómo restamos 4 a la posición, ya que la primera corresponde a 0.

Sin embargo, podemos también desplazarnos por las posiciones de un archivo para poder editar sus datos individualmente. Para ello, disponemos de la instrucción "SeekFile":

```
SeekFile Fichero, Desplazamiento
```

Vamos a añadir al ejemplo anterior las siguientes líneas:

```
SeekFile (Fichero,4)
Print "El numero de la posición
4 es: " + ReadInt(Fichero)
CloseFile(Fichero)
```

Aquí, nos situamos en la segunda posición, correspondiente a un desplazamiento de 4 bytes (números enteros), y leemos el número grabado en dicha posición (Ver "ejemplo2.bb").

Generalmente, no sabemos el número de datos que compone un fichero y necesitamos leerlo en su totalidad. Para ello, es necesario entrar en un bucle y leer todos los datos uno a uno. Para saber si hemos llegado al final del fichero disponemos de la instrucción "Eof". Supongamos que queremos imprimir todos los números enteros de un fichero:

```
Fichero = ReadFile ("Datos.dat")
While Not Eof ( Fichero )
    Print ReadInt (Fichero)
Wend
```

Como hemos podido comprobar, manipular los datos de un fichero es realmente sencillo. Vamos a ver a continuación la forma de leer o escribir esos datos en un archivo según al tipo a que pertenecen.

Para cada tipo de datos disponemos de una instrucción específica de lectura y escritura.

LEER Y ESCRIBIR DATOS NUMÉRICOS

Como sabemos, disponemos de cuatro tipos de datos numéricos en Blitz3D: Byte, Short, Int y Float.

En la página siguiente mostramos una tabla con las diferentes instrucciones para leer o escribir estos tipos de datos (Ver Cuadro 1).

LEER Y ESCRIBIR DATOS ALFANUMÉRICOS

Podemos leer o escribir una variable alfanumérica (string) o una línea completa de texto. Esto último es ideal para trabajar con documentos de texto (Ver Cuadro 2).

RELACIÓN ENTRE LOS FICHEROS Y LOS BANCOS DE MEMORIA

La utilización de bancos de memoria es ideal para realizar

NOTA

Las posiciones dentro de un fichero corresponden a los bytes que ocupan los datos contenidos en él. Por ejemplo, un número entero tiene 4 bytes, así que ocupará 4 posiciones en el fichero. Si tuviéramos 2 números enteros, el primero comenzaría en la posición 0 y el segundo en la 4.

TRUCO

Para calcular el desplazamiento necesario para averiguar la posición de un elemento dentro de un fichero podemos utilizar la siguiente fórmula: Desplazamiento = Elemento * Tamaño del elemento - Tamaño del elemento.

procesos de empaquetamiento o encriptación de datos debido a la gran velocidad de transferencia de datos que permite. Blitz3D nos proporciona dos funciones que permiten leer o escribir directamente de un fichero a un banco de memoria: "ReadBytes" y "WriteBytes".

"WriteBytes" permite escribir datos de un banco de memoria a un fichero. El fichero debe ser abierto con "WriteFile". Después de las operaciones de escritura debe ser cerrado con "CloseFile".

```
WriteBytes Banco de memoria,
Fichero, Offset, Número de
bytes a leer
```

En el siguiente ejemplo, creamos un banco de memoria para llenarlo de bytes y números enteros (10 de cada uno) Posteriormente, abrimos un fichero para introducir estos números en él con "WriteBytes". Para finalizar, cerramos el fichero y borramos el banco de memoria:

```
Banco=CreateBank(100)
For Offset = 1 To 10
 PokeByte  Banco,Offset,Rnd(100)
 PokeInt Banco, Offset + 1,
 Rnd(-10,10)
Next
; Abrimos el fichero
Fichero=WriteFile ("Fichero.bnk")
WriteBytes Banco, Fichero, 0, 10
CloseFile Fichero
FreeBank Banco
```

Una vez que tenemos los números del banco de memoria en un fichero, podemos leerlos posteriormente e introducirlos en otro banco:

```
Banco=CreateBank(100)
; Abrimos el fichero
Fichero=OpenFile ("Fichero.bnk")
ReadBytes Banco, Fichero, 0, 10
CloseFile Fichero
For Offset = 1 To 10
 PeekByte  Banco, Offset
 PeekInt   Banco, Offset + 1
Next
```

Código 1. Grabando posiciones

```
; GRABACIÓN
If Offset>Memoria-8
  SW_Grab=False
  Fichero_Grabacion=WriteFile("Camara.dat")
  WriteBytes BankMemX,Fichero_Grabacion,0,Memoria
  WriteBytes BankMemY,Fichero_Grabacion,0,Memoria
  WriteBytes BankMemZ,Fichero_Grabacion,0,Memoria
  WriteBytes BankMemPitch,Fichero_Grabacion,0,Memoria
  WriteBytes BankMemYaw,Fichero_Grabacion,0,Memoria
  CloseFile Fichero_Grabacion
EndIf
```

Código 2. Reproducir el fichero

```
;Activar Reproducción
If KeyDown(57) And SW_Play=False
  SW_Play=True
  Offset=0
EndIf
If KeyDown(56) And SW_Play=False
  SW_Play=True
  Offset=0
  Fichero_Grabacion=OpenFile("Camara.dat")
  ReadBytes BankMemX,Fichero_Grabacion,0,Memoria
  ReadBytes BankMemY,Fichero_Grabacion,0,Memoria
  ReadBytes BankMemZ,Fichero_Grabacion,0,Memoria
  ReadBytes BankMemPitch,Fichero_Grabacion,0,Memoria
  ReadBytes BankMemYaw,Fichero_Grabacion,0,Memoria
  CloseFile Fichero_Grabacion
EndIf
```

Estos procedimientos se pueden utilizar en el sistema de grabación de posición de la cámara que desarrollamos en el tutorial del número 15. Podríamos añadir las siguientes líneas de código cuando estamos grabando posiciones y terminamos (Ver Código 1).

De esta forma hemos creado un fichero ("Camara.dat") con todas las posiciones contenidas en los bancos de memoria. Ahora solo queda añadir el código para leer el fichero y reproducirlo. Por ejemplo, al pulsar la tecla "Alt" (Ver Código 2).

Cuadro 1. Leer y escribir datos numéricos

Datos numéricos	Leer	Escribir
Byte (0 – 255)	**ReadByte** (Fichero)	**WriteByte** (Fichero)
Short (0-65535)	**ReadShort** (Fichero)	**WriteShort** (Fichero)
Int (-2147483648 – 2147483648)	**ReadInt** (Fichero)	**WriteInt** (Fichero)
Float (-3.4 x 10 ^ -38 - 3.4 x 10 ^ -38)	**ReadFloat** (Fichero)	**WriteFloat** (Fichero)

Cuadro 2. Leer y escribir datos alfanuméricos

Datos alfanuméricos	Leer	Escribir
String (variable alfanumérica)	**ReadString** (Fichero)	**WriteString** (Fichero)
Line (línea de texto)	**ReadLine** (Fichero)	**WriteLine** (Fichero)

En el próximo número...

... aprenderemos a utilizar el audio en nuestros programas y a manipular archivos de vídeo.

Realización de un vídeo para nuestro juego (I)

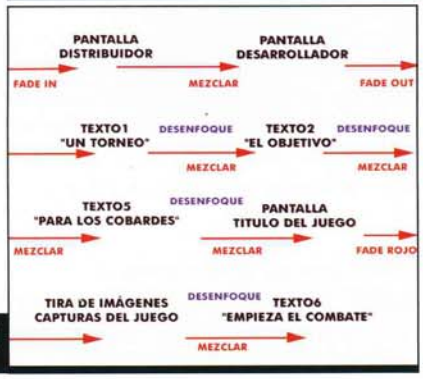
Con esta entrega vamos a desarrollar un pequeño vídeo que servirá para presentar nuestro juego.

Para ello utilizaremos el editor de vídeo de Ulead MediaStudio Pro 6.5 y Paint Shop Pro.

LA IDEA

Se puede conseguir una presentación digna con poco esfuerzo, solo es cuestión de tener las ideas claras de lo que se quiere hacer. Para nuestra presentación utilizaremos los logotipos del desarrollador, distribuidor y del juego. También utilizaremos algunas pantallas de texto y capturas del juego. La idea es mostrar poco a poco en pantalla algunas frases clave que desvelen el argumento del juego y que den paso a una secuencia de capturas. Antes de empezar es importante realizar un guión del vídeo para saber en todo momento cómo se desarrollará la presentación (Ver Fig. 1).

Para desarrollar las pantallas de texto vamos a utilizar Paint Shop Pro.



Guión básico del vídeo de presentación, gracias al cual sabremos en todo momento cómo se desarrollará.

DIBUJANDO LAS PANTALLAS DE TEXTO

Dentro del programa creamos un documento nuevo con fondo negro y de 800 x 600 de resolución. Seguidamente, elegimos la herramienta de texto para escribir la primera de las frases que aparecerá en pantalla. En la ventana de texto *Text Entry* escribimos la frase "UN TORNEO" con el tipo de letra que diseñamos para "Zone of Fighters" ("zof.ttf", "dungeon.ttf" modificada). Si no disponemos de la fuente se puede encontrar en el directorio "Extras" del CD.

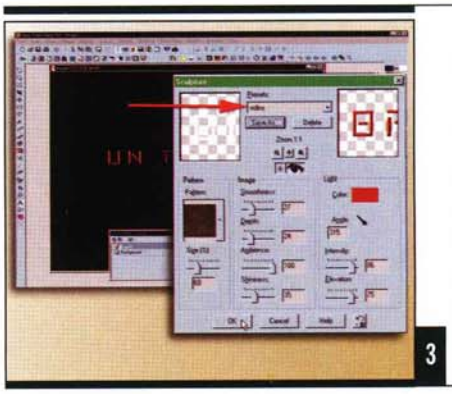
Seleccionamos un tamaño de letra de 48 y un color de relleno *Styles / Fill* blanco. Los "Strokes" de *Styles* y *Textures* los anulamos, así como el "Fill" de *Textures* (Ver. Fig. 2).

Al aceptar hemos creado una nueva capa vectorial, la cual debemos rasterizar (convertir a mapa de bits) para poder aplicarle efectos de imagen. Una vez rasterizada le aplicaremos un efecto de textura para cambiar el aspecto de las letras. En el menú *Effects* elegimos la opción *Sculpture* en *Texture Effects*. Y aplicamos los siguientes parámetros:

En *Pattern* seleccionamos el patrón número 21 (Leather.bmp) con un tamaño del 68%. En *Image* aplicamos un *Smoothness* de 27, un *Depth* de 24, un *Ambience* de 100 y un *Shininess* de 35. En la sección *Light* seleccionamos un ángulo *Angle* de 315, una *Intensity* del 86% y una *Elevation* de 75. Pulsamos en *Save As* ... para grabar nuestro efecto y poder así utilizarlo para los demás textos. Asignemos el nombre "vídeo" al nuevo *preset* (Ver Fig. 3).

Para escribir el texto de las frases utilizaremos la fuente del juego plana y de color blanco.

Para el resto del texto es solo cuestión de escribirlos y aplicarles el nuevo efecto que hemos creado. Creamos una nueva capa para cada frase. Luego debemos grabar una imagen con cada una de ellas. Para ello, ocultamos las capas que no nos interesan y colocamos la frase, con la herramienta de transformación, más o menos en el centro



Un efecto de textura tipo "Sculpture" es ideal para obtener la apariencia deseada para las letras.

4

Una vez que tengamos todas las capas, grabamos una pantalla con cada frase. Para ello, mantenemos visible la que nos interesa.

de la pantalla. Salvamos en formato .bmp en *File/ Save As*. Seguiremos una secuencia de nombres, por ejemplo, "Texto1.bmp" para la frase "UN TORNEO", "Texto2.bmp" para "EL OBJETIVO" y así sucesivamente. En total debemos tener 6 pantallas (Ver Fig. 4):

"Texto1.bmp" ➡ "UN TORNEO"
"Texto2.bmp" ➡ "EL OBJETIVO"
"Texto3.bmp" ➡ "SOBREVIVIR"
"Texto4.bmp" ➡ "NO HAY PIEDAD"
"Texto5.bmp" ➡ "PARA LOS COBARDES"
"Texto6.bmp" ➡ "EMPIEZA EL COMBATE"

OBTENIENDO LAS CAPTURAS DEL JUEGO

La siguiente fase de nuestro vídeo es adquirir, desde el juego, las capturas que nos servirán para terminar nuestra presentación. Para ello, solamente tenemos que pulsar, durante una partida, la tecla "S". En ese instante el búfer de pantalla se grabará en un fichero de mapa de bits (.bmp) en el directorio del juego ("C: \ zone of fighters\"). Simplemente, se ha añadido al código del juego la siguiente función para poder realizar las capturas, la cual es llamada continuamente desde el bucle principal de la partida (Ver Código 1)

EL EDITOR DE VÍDEO

Una vez que tenemos todo el material gráfico pasamos al editor para conformar el vídeo. Hemos elegido Ulead MediaStudio porque posee un editor de vídeo muy sencillo de utilizar y a la vez potente. Suficiente para nuestras necesidades. No vamos a profundizar en todas las posibilidades que nos ofrece, pero sí aprenderemos lo fundamental para el desarrollo de cualquier presentación que se nos ocurra (Ver Fig. 5).

Al ejecutar la aplicación y antes de entrar en el editor, debemos seleccionar qué tipo de vídeo vamos a crear. Para ello, seleccionamos, de momento, la plantilla *Presentation Video* (25 fps, Indeo) de la lista de plantillas *Existing project templates*. Ya dentro del editor se nos presentan cuatro ventanas: *Timeline, Preview, Source* y *Production Library*.

En *TimeLine* es donde editamos nuestro vídeo, en *Preview* se muestra un previo de la edición, en *Source* se visualiza el resultado final y *Production Library* es un lista de todos los efectos disponibles para la producción audiovisual (Ver Fig. 6).

La versión *trial* de este programa mostrará continuamente en la pantalla de previo y en el vídeo una "X" de color rojo, aun así podemos ver cómo resulta nuestra pequeña producción.

5

Antes de comenzar a editar con Ulead MediaStudio es necesario elegir una plantilla para el proyecto.

6

Ventana principal del editor de Ulead MediaStudio Pro 6.5, con sus cuatro áreas principales: previo, fichero, efectos y el montaje propiamiente dicho.

Código 1. Función para obtener las capturas

```
Function Salvapantalla()
  ; salva pantallas con "S"
  If KeyDown (31)
    While fileExists("c:\zone of fighters\"+captura+".bmp")
      captura=captura+1
    Wend
    SaveBuffer (FrontBuffer(),"c:\zone of fighters\"+captura+".bmp")
    captura=captura+1
  EndIf
End Function
```

En el próximo número...

... entraremos de lleno en la edición del vídeo de presentación de nuestro juego.

Simuladores de vuelo

En esta entrega empezamos a recorrer la historia del género de la simulación con los simuladores de vuelo para PC.

Los simuladores de vuelo han estado siempre ligados al sector civil y militar de la aviación por los grandes beneficios que reportan en la detección de errores o en cursos de entrenamiento para pilotos. Al principio, solo los grandes ordenadores podían simular el comportamiento de un aeroplano con más o menos calidad. La llegada a los hogares de los microordenadores supuso, una vez más, un revulsivo para el desarrollo de simples simuladores para uso lúdico. La producción de juegos para ordenador basados en este género llevó al desarrollo de simuladores de muy diversa índole, buscando, cada vez más, la acción propia de un juego que la estricta aplicación de la simulación. Así, podemos encontrar simuladores aeronáuticos de aviones comerciales, de combate o de helicópteros. También, traspasando la frontera de lo real, el género se traslada a la ciencia-ficción, apareciendo simuladores espaciales.

SIMULADORES DE VUELO CIVILES

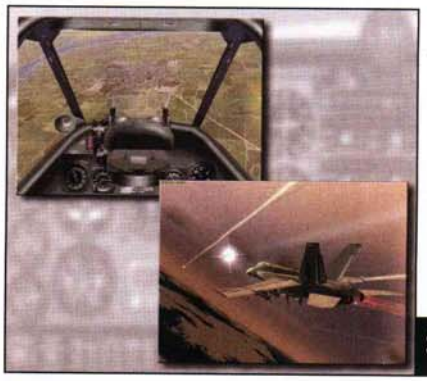
Básicamente, este tipo de simuladores ha creado el origen de este género para ordenadores personales. El concepto primordial es simular el comportamiento de aviones comerciales a través de gráficas 3D y control gráfico de instrumentales de navegación representado en pantalla. El primer juego basado en esta idea fue *Flight Simulator*. Fue desarrollado por Bruce ArtWick en 1979 para ordenadores Apple II después de fundar, con *Stu Moment*, la compañía SubLogic. Estaba basado en su tesis para la Universidad sobre la representación gráfica en tiempo real de una simulación de vuelo. Poseía gráficos en 4 colores y un panel de control con dos instrumentos de medida. No fue hasta 1982 cuando se desarrolla la versión para IBM PC. Editada por Microsoft, poseía 4 colores, gráficos vectoriales en 3D y hasta 8 instrumentos de medida. Además incorporaba climatología, 9 vistas diferentes y 4 escenarios con 20 aeropuertos. Todo un lujo en aquella época. El juego se convirtió en todo un éxito, hasta el punto que sus usuarios compraban mejores ordenadores solo para jugarlo. Durante los años 1984 y 1987 se realizaron 14 versiones para diferentes sistemas. Con la versión *Microsoft Flight Simulator 98* se incluyó la posibilidad de manejar helicópteros. El juego cumplía por entonces 15 años de existencia y ya había vendido la friolera de 10 millones de copias. Actualmente, *Flight Simulator* va por la octava generación con *Flight Simulator 2002* versión 8.0 y está en el libro Guinness de los récords al juego más vendido de la Historia con más de 25 millones de copias. Hoy día, *Flight Simulator* es un juego de culto en su género e insuperable y es utilizado en el sector aeronáutico como simulador de entrenamiento de pilotos. Al margen de *Flight Simulator*, otra desarrolladora compite con Microsoft en los simuladores civiles con la serie *Flight Unlimited*, nos referimos a Looking Glass Studios. *Flight Unlimited* se caracteriza por poseer gráficos de escenarios más realistas, con efectos especiales increíbles y una forma más entretenida de juego. Con *Flight Unlimited III* todos los escenarios están generados a partir de datos de satélites.

1

Flight Simulator, el primer simulador de vuelo y el más vendido de la historia.

2

F15 Strike Eagle (arriba) y GunShip (abajo), entre los precursores de los actuales simuladores de vuelo de combate.



3

European Air War (arriba): máximo exponente de los simuladores de la Segunda Guerra Mundial. F/A-18 Simulator de Jane´s Combat Simulations.

4

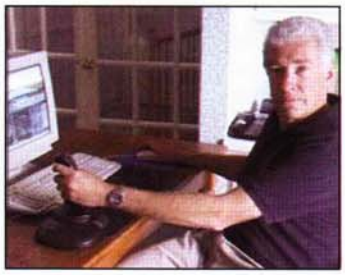
Falcon 4.0 (arriba) y Comanche 4 (abajo) forman parte de la serie de simuladores de vuelo de combate más populares.

LA BIOGRAFÍA...

BRUCE ARTWICK

Creador de **Flight Simulator**

Entró en la universidad para estudiar ingeniería de computadoras en 1973 pero cambió a ingeniería eléctrica; aun así, siempre estuvo interesado en la aviación. Llegó a ser piloto mientras desarrollaba una técnica para visualizar mediante gráficos en 3D un vuelo simulado. Partiendo de esos estudios desarrolló el primer simulador de vuelo en 3D de la historia, *Flight Simulator* para Apple II, totalmente programado en código máquina. Llegó a fundar dos compañías de desarrollo de simuladores: SubLogic y BAO Ltd., que han desarrollado toda la serie *Flight Simulator* para todos los sistemas. Actualmente, BAO pertenece a Microsoft y SubLOGIC a Sierra, pero ArtWick sigue escribiendo libros y creando los más complejos simuladores para PC.



Bruce ArtWick

SIMULADORES DE VUELO DE COMBATE

Otra alternativa de simulación propuesta y que atiende más a la jugabilidad es la unión entre los aspectos aeronáuticos y la acción del combate. Casi toda la producción de simuladores de vuelos atiende a esta premisa, buscando no solo una simulación aérea, sino simular misiones de combates históricas o ficticias. Prácticamente, a lo largo de la historia, desde los primeros títulos de 8 bits como *Kennedy Approach* o *MiG Alley Ace* (primer simulador de combate multijugador en pantalla dividida), las diferencias entre los títulos publicados se basaban en la calidad de la representación gráfica de los combates. De estos primeros títulos, precursores de los simuladores de hoy, destacamos la serie *F-15 Strike Eagle, Spitfire Ace* o *Gunship* y *F-19 Stealth Fighter* de Microprose. Los simuladores actuales son exquisitos gráficamente y cada vez se desarrollan más títulos de calidad. Microsoft aparece en este tipo de simulación con *Combat Flight Simulator* 1 y 2. Pero, en virtud a sus cualidades, matizamos la fantástica producción *European Air War* de Microprose con un realismo aplastante con hasta 256 aviones simultáneos en combate o el original manejo de bombarderos B-17 en *B-17 Flying Fortress*. Este género siempre ha estado acompañado por cierta dificultad en el manejo del juego y que no ayudaba mucho a capturar adeptos ajenos a la simulación aérea. Sin embargo, algunas producciones se decantaban por una simulación aceptable, orientada a la acción inmediata y sin muchos entresijos en el control de los aviones. Desarrolladoras como Jane´s Combat Simulations optaron por esta fórmula y produjeron títulos de gran calidad como la serie *USAF* con gráficos detallados al milímetro de aviones y escenarios. Pero es en producciones de nivel superior orientada a pilotos expertos, donde encontramos títulos realmente sorprendentes como *EF2000* y de nuevo las producciones de Jane´s Combat Simulations como la serie *SkunkWorks, F-15* o el fantástico simulador del caza *F/A-18 Hornet, F/A-18 Simulator*. No podemos olvidar la saga *Falcon 4.0* de Micorprose con su exquisita IA y sus campañas en tiempo real. Tampoco queremos terminar sin mencionar la famosa saga de simuladores para helicópteros *Comanche* de Novalogic con más de 3 millones de copias vendidas.

SIMULADORES ON-LINE

Desde que internet llega a los hogares de todo el mundo, los simuladores de vuelo han entrado a formar parte también del juego On-Line. Al igual que ocurre con géneros como el FPS táctico o el RPG, aparecen títulos en donde el jugador participa en batallas aéreas ficticias o históricas. El primer simulador On-Line fue *Air Warrior*. Le siguió *War Birds* y actualmente destaca *Aces High* en el que también es posible manejar otros vehículos militares.

SIMULADORES DE VUELO ESPACIALES

Paralelo al desarrollo de simuladores de vuelos reales, aparecieron otros basados en el vuelo y combate espacial. El liderazgo de este tipo de simulación desde el comienzo de la década de los 90 ha sido para la serie *Wing Commander* (Origin) y *X-Wing* (Totally Games) El primero destacó por la inclusión de escenas narrativas en 3D durante el juego para dar más énfasis al guión. Por otro lado, *X-Wing* se basaba en la laureada serie de películas *Star Wars*, lo que le proporcionaba un éxito asegurado. Sin embargo, otros títulos aparecieron para intentar hacer sombra a estas series a partir de 1998.

En el próximo número...

... hablaremos de los simuladores deportivos.

Cuestionario Videojuegos 16

Preguntas

1. Escribe las sentencias necesarias para leer y mostrar los ficheros de un directorio.
2. Escribe el código para abrir un fichero y grabar en él 10 números enteros aleatorios.
3. ¿Cómo una entidad puede detectar la proximidad de otra?
4. ¿Por qué es necesario incluir todas las variables que intervienen en el comportamiento de una entidad (contadores, switchers,etc.) en su estructura de tipo?
5. ¿Qué dos formas podemos usar para disponer de un terreno en nuestro juego?
6. Si usamos Paint Shop Pro para dibujar el mapa de alturas, ¿cómo podemos visualizar el terreno resultante?
7. ¿Cómo podemos importar un fichero de audio en formato .WAV en Anvil Studio?
8. ¿Cuáles son los procedimientos a seguir para añadir una muestra a una pista de ritmos?
9. ¿Cómo podemos obtener capturas de nuestro juego?
10. Antes de empezar a editar con Ulead MediaStudio, ¿qué es lo primero que debemos hacer?

Respuestas al cuestionario 15

▷ **1.** While Not KeyHit (16)
```
    PROCESOS
Wend
```

▷ **2.** Mediante la instrucción MouseZ().
El valor devuelto por MouseZ() se incrementa cuando la rueda se desliza hacia delante y decrementa cuando lo hace hacia atrás.

▷ **3.** Asignando coordenadas X y Z aleatorias dentro de un rango determinado:
```
CoordX= Rand (RangoX1, RangoX2)
CoordZ= Rand (RangoZ1, RangoZ2)
PositionEntity Entidad, CoordX, Altura Terreno, CoordY
```

▷ **4.** Comprobando que todos los datos leídos tienen valor 0 o "". O bien utilizando la instrucción Eof:
While Not Eof (Fichero)

▷ **5.** Para crear un cubo en Bryce, solo tenemos que pulsar en el icono con el dibujo de un cubo situado en el menú *Create*. Para situarlo frente a la cámara podemos hacerlo con las herramientas de edición del menú *Edit* desplazando el cubo, o bien moviendo la cámara con los iconos de navegación.

▷ **6.** Para asignar y editar una textura a un modelo en Bryce debemos entrar en el laboratorio de materiales pulsando sobre la letra "M" situada en la parte inferior derecha del modelo.

▷ **7.** Para seleccionar un grupo de notas en la sección compose de Anvil Studio debemos hacerlo sobre la línea de tiempo.

▷ **8.** En primer lugar elegimos la calidad de la muestra y posteriormente pulsamos sobre el botón "VU". Para terminar hacemos *Play* en el CD y pulsamos el botón "REC".

▷ **9.** Grabando los datos en matrices o bien utilizando bancos de memoria.

▷ **10.** Utilizando bancos de memoria y usando las instrucciones:
PokeByte, PokeShort, PokeInt o PokeFloat para escribir en memoria y:
PeekByte, PeekShort, PeekInt o PeekFloat para leer.

Contenido CD-ROM 16

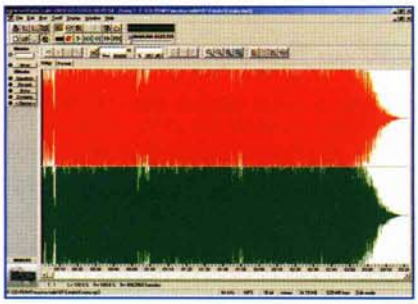
▶ AUDIO

■ Audiotest
Chequea rápidamente el audio de tu PC antes de empezar a editar y manipular ficheros de audio.

■ Mkw Audio Compression Tool
Ahorra espacio comprimiendo tus archivos de audio.

■ Total Recorder
Graba sonidos y música usando esta aplicación, que viene junto a los drivers necesarios.

■ WavePurity 4.10

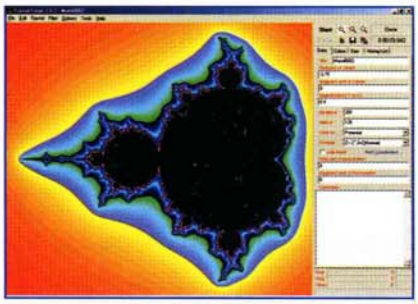


Editor muy completo de audio que te será de gran ayuda para tus ficheros de música.

▶ DISEÑO 2D

■ Antares 8
Avanzado navegador y visualizador de imágenes diseñado específicamente para manejarlas cómoda y fácilmente.

■ FractalForge



Aprende a dibujar fractales que podrás utilizar para tus fondos y paisajes psicodélicos.

■ ImageKeep Express 1.0.1
Crea bases de datos con tus imágenes y así tenlas siempre ordenadas y a mano.

■ Magic Converter 3.0
Con esta utilísima aplicación podrás cambiarle el formato a numerosos grupos de imágenes.

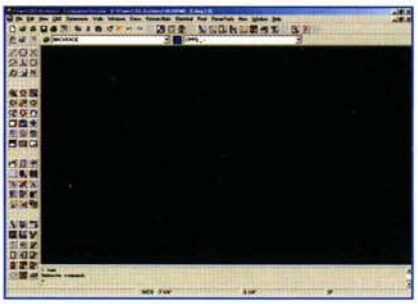
■ PFS Manager 2.05
Maneja tus álbumes fotográficos y ten ordenadas tus imágenes en sus categorías correspondientes.

▶ DISEÑO 3D

■ 3D Coder
Rápida aplicación para realizar pequeños ajustes en las imágenes en tres dimensiones.

■ 3D RealTime Visualizer 1.0
Crea objetos en 3D rápidamente y sin esfuerzo.

■ PowerCAD Architect



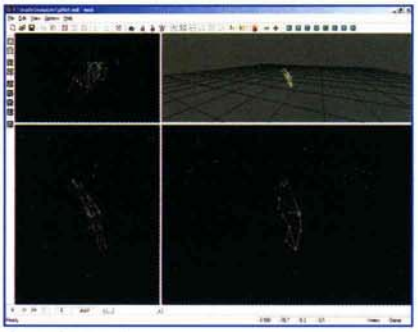
Magnífico programa de arquitectura para crear interiores en 3D, edificios y todo lo que se te ocurra.

■ Wings 3D
Sencillo editor de imágenes en tres dimensiones con una interfaz muy intuitiva.

▶ PROGRAMACIÓN

■ ActiveZipper Pro 1.0
Herramienta para Visual Basic para comprimir y descomprimir fácilmente.

■ 3D GameStudio A4



Completa suite para programar y renderizar juegos en 3D.

■ Pocket C
Editor de código "de bolsillo", para ver rápidamente cómo van tus líneas de código.

▶ JUEGOS

■ Falcon 4.0
Simulador de combates aéreos con un gran concepto de la inteligencia artificial y con campañas en tiempo real.

■ FreeSpace
Simulador de vuelos espaciales, surca los cielos gracias a este excelente juego.

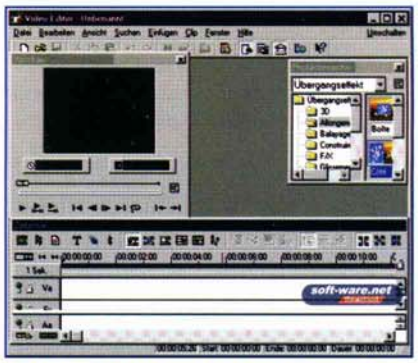
■ X-Plane 6.4

Magnífico simulador de vuelo con el que te creerás pilotando de verdad.

■ Zone of fighters
Como todas las semanas, nuestro juego tal y como va quedando.

▶ VÍDEO

■ Easy Video Joiner 4.01
Con esta útil aplicación podrás unir fácilmente varios clips de película en uno solo.



■ Ulead Media Studio Pro 6.5
Nueva oportunidad de conseguir este excelente software.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.